POLITECNICO DI TORINO | MSc ARCHITECTURE FOR SUSTAINABILITY | 2024-2025

MULTI-MODEL AI SOLUTIONS FOR WINDOW CONFIGURATION IN COMPLEX ARCHITECTURAL SCENARIOS WITH LIMITED DATA

Supervisor: Lo Turco Massimiliano, Lo Verso Valerio Roberto Maria, Tomalini Andrea
Student: Meng Shisen

ARCHITECTURE FOR SUSTANABILITY

ABSTRACT

This study combines custom rules with multiple deep learning predictive models to develop a program that rapidly outputs optimal window configurations meeting lighting requirements based on input building features. The 2D building layout images are first converted into corresponding textual feature information. The designed programme can quickly predict the optimal window configuration after reading this information. At the same time, it can specify the number of windows to be opened in a room and the orientation of the windows by limiting the amount of input data. It solves the problem with most of the current automatic building layout generation models that the generated layouts lack window information and are not suitable for actual design, reducing their reference value in terms of building performance. The challenges encountered in this study include how to quickly collect a sufficient number of compliant and highly accurate building layout datasets using professional lighting simulation software, and how to train the prediction model with a small number of datasets so that it can accurately predict window configurations under highly complex building conditions. To address these issues, this study created a program on the Grasshopper platform (a visual programming plug-in for the 3D modelling software Rhino) for performing automated lighting simulations and acquiring high-precision simulation datasets. At the same time, by combining custom rules with multiple deep learning predictive models, the program achieves accurate predictions of window configurations under complex architectural conditions, rather than relying on a single predictive model. This approach significantly improves prediction accuracy and demonstrates the ability to handle complex architectural conditions, even with a small dataset.

Keywords: Automated building layout generation, Optimal Window Size, Window-to-wall ratio (WWR), Multi-objective optimization, RPLAN Dataset, Lighting simulation, Deep Learning Predictive Modeling, Multi-Model AI Solutions

INTRODUCTION

Space layout design is a critical phase in architectural design, and automatic space layout generation has shown great potential in the design process. Significant progress has been made in automated layout generation using deep learning, especially in the automatic design of residential floor plans. Many deep learning models excel in accuracy and diversity, accuracy in the sense that, when inputting constraints, the model generates building plans that are correct and essentially contain that information based on those constraints, and diversity in the sense that, when fine-tuning the input constraints each time, the model generates a different building plan that isn't a simple search-and-replicate of multiple datasets.

However, these models have many shortcomings in practical applications. One of the major problems is that the generated floor plans of homes often lack window information, which requires users to manually configure the number and size of windows in each room according to their preferences. From the perspective of the user community, this leads to two main consequences. For users without architectural expertise, such floor plans lacking windows are difficult to use directly. Furthermore, due to the lack of expertise, manually configured window layouts may not result in optimal lighting. On the other hand, for users with architectural expertise, these models can often be used as a reference for the initial floor plan design. If users are provided with a floor plan from the start that already contains a correctly configured window layout, they can effectively avoid the hassle of having to repeatedly adjust the building layout to solve lighting problems later on.

There are a number of ways to configure windows in a building plan, which can generally be categorised as manual configuration by professionals, using lighting simulation software and using machine learning predictive models.

Each of these methods has its own advantages and disadvantages. First, having professionals manually design the window configuration of building plans can indeed ensure that the configured layout avoids basic errors. However, its disadvantage is that it is difficult to find enough professionals with rich architectural knowledge. In addition, the manual configuration method cannot efficiently process a large number of building plans at the same time.

Second, using lighting simulation software offers the advantage of obtaining lighting results calculated with precision. However, when applied to a large number of building layouts or complex designs, it requires a significant amount of time to process.

Lastly, employing machine learning prediction models enables results to be generated at a very high speed and allows for the continuous processing of large volumes of building floor plans. The disadvantage, however, is that the predictive models need to be retrained for each condition as building performance is influenced by factors such as environment and materials. Retraining such models for complex building conditions requires a large and accurate dataset and redesign of the model structure. Both data collection and model design are time-consuming and there is no guarantee that the resulting models will be highly accurate.

Based on the limitation that current automatically generated building layouts lack window configurations, we chose to use a deep learning predictive model. This approach can be directly integrated with such layout generation models. After generating building layouts, relevant building information can be extracted directly from these layouts to predict the corresponding window configurations, which can then be added directly to the layout.

Our study consists of the following steps. First, the RPLAN dataset (a large image dataset) was processed by converting the image data into JSON files containing detailed building information. In this step, additional code was implemented to successfully extract more comprehensive building details. These details are crucial for subsequent model visualisation, multi-objective optimisation tasks and predictive model training.

The JSON file is then imported into the Grasshopper platform of the Rhino software to generate the corresponding 3D model. These 3D models can be combined with the ClimateStudio lighting simulation plug-in and can perform lighting simulation. Then, we set multiple conflicting lighting indicators as optimization parameters for the multi-objective optimization software. When the multi-objective optimization software is running, it will continuously change the size of each window in the window room until the lighting balance of the room reaches the optimal effect. After the calculation is completed, we will use a custom table to record the final results. These recorded window size parameters and building layout information will be used to be the new dataset.

During the training phase of the deep learning prediction model. In our initial attempts to use a single model to predict the window size, we found that a single model did not perform well on complex, small datasets. In our research, after iterative adjustments, we used an approach that combines custom rules with multiple deep learning predictive models to predict the optimal window size for the input building layout. The accuracy of this approach was significantly improved.

Our research contributions can be summarized as follows:
1. A method was developed to rapidly and automatically visualize a building floor plan dataset and perform multi-objective optimization to obtain window configuration data that achieves optimal lighting performance. Raw graphical floor plan data is challenging to use directly for lighting simulations and multi-objective optimization. Therefore, we first converted the graphical data into textual information, which was subsequently visualized and imported into

lighting simulation software. Additionally, a method combining the minimum window size with minimal window size variation parameters was proposed for multi-objective optimization. This approach significantly reduced the runtime of the optimization process and avoided unnecessary computations caused by minor variations in window parameters.

2. We first tried to use a single model to predict the size of each window in the room, but we found that the single model did not perform well on limited datasets. And it could not predict complex changes in building layouts. After many attempts, we decided to build a predict program. This program combine custom rules and multiple deep learning prediction models to predict the optimal window size for the input building layout. After validating the accurary. We found that this method significantly improved the prediction accuracy.

Finally, the figure below(Figure 1) is the basic framework of our entire research.

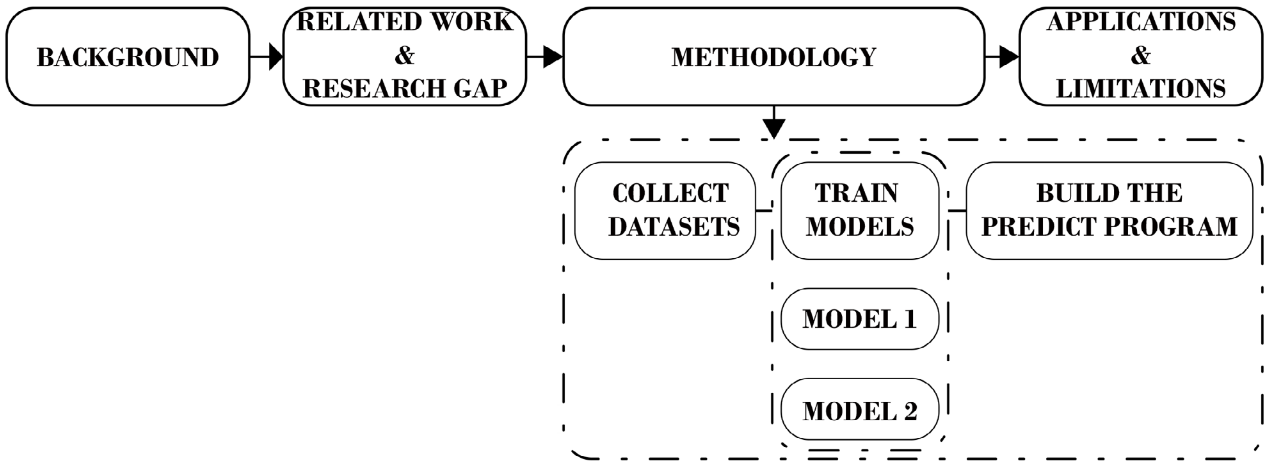

BACKGROUND
RELATED WORK & RESEARCH GAP
METHODOLOGY
APPLICATIONS & LIMITATIONS
COLLECT DATASETS
TRAIN MODELS
BUILD THE PREDICT PROGRAM
MODEL 1
MODEL 2

Figure 1. Research Framework (drawn by the author)

INDEX

Part l Background

1.1 Related Concepts

1.1.1 Machine Learning

Machine learning is typically classified into supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning, alongside approaches such as deep learning, transfer learning, and ensemble learning[1]. Our research focuses primarily on supervised learning, which involves learning a function that maps inputs to outputs based on labeled data. This process uses training examples to infer a function that fits the given input-output pairs[2]. Supervised learning is applied when there are specific objectives to achieve based on a predefined set of inputs[3]. In contrast to supervised learning, unsupervised learning does not require labeled datasets and does not require human intervention, i.e. a data-driven process[2]. This is widely used for extracting generative features, identifying meaningful trends and structures, groupings in results, and exploratory purposes. The most common unsupervised learning tasks include clustering, density estimation, feature learning, dimensionality reduction, finding association rules, anomaly detection, etc.

1.1.2 Generative Artificial Intelligence (AI)

Generative artificial intelligence (GenAI) refers to artificial intelligence models that generate high-quality text, images, and other content based on various types of input data [4]. One of the most important features of generative artificial intelligence (GenAI) is that it can input multimodal data. It is able to generate output from multiple types of data input (such as text, speech, audio, video, and images) [5]. GenAI can operate in an unsupervised or semi-supervised framework and generate new results by changing the input distribution [6]. This ability is very important in applications such as image and video generation, sequence modeling, and speech enhancement [7].

Complex generative models are usually trained on large datasets using clue-based methods, which leads to poor learning [8] or even zero learning [9]. However, combining reinforcement learning with human feedback (RLHF) with these models (such as ChatGPT) has significantly improved the learning effect of the model [10].

Our research focuses on generative adversarial networks (GANs). It was first proposed by Ian Goodfellow in 2014. The main principle of this technology is to convert noise vectors into samples through a single generative process. In GAN, the two main parts are the generator and the discriminator. The task of the generator is to generate images from random noise, while the task of the discriminator is to identify the similarity between the images generated by the generator and the real images. Once the discriminator determines that the image generated by the generator is fake, the generator will be improved again, generate new images, and input them into the discriminator for judgment again [11].

1. Mohammed, Mohssen & Khan, Muhammad & Bashier, Eihab. (2016). Machine Learning: Algorithms and Applications. 10.1201/9781315371658.

2. Han, Jiawei & Kamber, M. & Pei, J.. (2006). Data Mining: Concepts and Techniques. 585-631.

3. Sarker, Iqbal & Kayes, A. S. M. & Badsha, Shahriar & Alqahtani, Hamed & Watters, Paul & Ng, Alex. (2020). Cybersecurity data science: an overview from machine learning perspective. Journal of Big Data. 7. 10.1186/s40537-020-00318-5.

4. Martineau, K. "What is Generative AI?" IBM Research, 2023. Retrieved from https://research.ibm.com/blog/what-is-generative-AI.

5. Cao, Yihan & Li, Siyu & Liu, Yixin & Yan, Zhiling & Dai, Yutong & Yu, Philip & Sun, Lichao. (2023). A Comprehensive Survey of AI-Generated Content (AIGC): A History of Generative AI from GAN to ChatGPT. 10.48550/arXiv.2303.04226.

6. Jovanovic, Mladjan & Campbell, Mark. (2022). Generative Artificial Intelligence: Trends and Prospects. Computer. 55. 107-112. 10.1109/MC.2022.3192720.

7. H. Cao et al., "A Survey on Generative Diffusion Models," in IEEE Transactions on Knowledge and Data Engineering, vol. 36, no. 7, pp. 2814-2830, July 2024, doi: 10.1109/TKDE.2024.3361474.

8. Liu, Pengfei & Yuan, Weizhe & Fu, Jinlan & Jiang, Zhengbao & Hayashi, Hiroaki & Neubig, Graham. (2022). Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ACM Computing Surveys. 55. 10.1145/3560815.

9. Wang, Yaqing & Yao, Quanming & Kwok, James & Ni, Lionel. (2020). Generalizing from a Few Examples: A Survey on Few-shot Learning. ACM Computing Surveys. 53. 1-34. 10.1145/3386252.

10. Ouyang, Long & Wu, Jeff & Jiang, Xu & Almeida, Diogo & Wainwright, Carroll & Mishkin, Pamela & Zhang, Chong & Agarwal, Sandhini & Slama, Katarina & Ray, Alex & Schulman, John & Hilton, Jacob & Kelton, Fraser & Miller, Luke & Simens, Maddie & Askell, Amanda & Welinder, Peter & Christiano, Paul & Leike, Jan & Lowe, Ryan. (2022). Training language models to follow instructions with human feedback. 10.48550/arXiv.2203.02155.

11. Goodfellow, Ian & Pouget-Abadie, Jean & Mirza, Mehdi & Xu, Bing & Warde-Farley, David & Ozair, Sherjil & Courville, Aaron & Bengio, Y.. (2014). Generative Adversarial Networks. Advances in Neural Information Processing Systems. 3. 10.1145/3422622.

Since the introduction of GANs, various models have been developed based on this approach. For example, Isola et al. proposed Pix2Pix in 2018, an algorithm that generates images conditioned on other images, known as image-to-image translation or conditional GAN (cGAN) [12].

1.1.3 Self-attention Mechanism

Self-attention (sometimes referred to as internal attention) is an attentional mechanism that relates different positions in a single sequence to compute a representation of that sequence [13]. In simple terms, self-attention can be understood as an algorithm that helps the model understand the important relationships between different parts of the input data. Its main function is to dynamically assign "attention weights" to each element in the data, thereby comparing all the elements in it to determine which parts are more important.

The most important feature of the self-attention mechanism is that it can pay attention to global associations. For example, in sentence translation, the connection between two words that are far apart can also be captured. This ability is particularly important for processing continuous data such as text or time series. Because of this feature, the self-attention mechanism has become a basic technology for modern algorithms such as Transformer.

Our research also takes advantage of this feature of the self-attention mechanism and introduces it into the model to improve the accuracy of the model when allocating different window sizes in the room.

1.1.4 The Relationship Between Window Configuration and Daylighting in Architecture

When designing a building plan, architects often consider the building environment to ensure that performance requirements such as ventilation, energy efficiency and lighting are met. The location and size of windows are particularly important in influencing the daylighting and energy performance of a building. In this study, we focus on daylighting in residential spaces, examining how different window configurations affect daylighting.

Much research has been conducted on architectural windows and lighting.Ignacio Acosta et al. evaluated visual comfort and energy savings by quantifying lighting metrics for various windows in residential rooms. Their results showed that daylight autonomy increased with the size of the glazing surface and the reflectivity of surfaces near the back of the room, while the effect near the façade was negligible. In addition, the relationship between energy consumption and window shape was not significant. Higher positioned windows resulted in higher illuminance at the back of the room compared to centrally positioned windows [14].

Similarly, Liu et al. analysed illuminance distribution and daylight glare for different window configurations. Their study showed that the maximum illuminance decreased by 2148 lx and the minimum illuminance gradually increased by 93 lx when the sill height was increased from 0.8 m to 1.6 m. Rectangular windows produced the most glare, while ribbon and arched windows produced the least. In addition, glass transmittance is an important factor. The higher the transmittance, the higher the luminance, but the more severe the glare. Therefore, careful consideration is needed in determining the appropriate glass transmittance [15].

12. Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A.A. "Image-to-Image Translation with Conditional Adversarial Networks." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, 1125–1134. http://dx.doi.org/10.48550/arXiv.1611.07004.

13. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. Adv Neural Inf Process Syst 2017;30. https://doi.org/10.48550/ arXiv.1706.03762.

14. Acosta García, I.J., Campano, M.Á. y Molina Rozalem, J.F. (2016). Window design in architecture: Analysis of energy savings for lighting and visual comfort in residential spaces. Applied Energy, 168, 493-506. https://doi.org/10.1016/j.apenergy.2016.02.005.

15. Liu, X., Sun, Y., Wei, S., Meng, L., and Cao, G. "Illumination Distribution and Daylight Glare Evaluation within Different Windows for Comfortable Lighting." Results in Optics 3 (2021): 100080. https://doi.org/10.1016/j.rio.2021.100080.

1.1.5 Multi-Objective Optimization

Multi-objective optimisation requires balancing multiple conflicting objectives. Multi-objective evolutionary algorithms (MOEA) have become an important method for solving balancing problems in evolutionary computation. Due to conflicts between objectives, multi-objective optimisation problems (MOPs) usually do not have a single optimal solution. Instead, they produce a set of Pareto-optimal (P-O) solutions that balance the objectives. The classical approach to solving MOPs is to transform them into single-objective problems using aggregation functions to obtain P-O solutions. Commonly used techniques include the weighted summation method, the Tchebycheff method, and the Min-Max method, but these methods have limitations in correctly assigning weights and dealing with concave Pareto bounds (PFs). On the contrary, evolutionary algorithms (EA) can generate approximate solutions that satisfy practical requirements [16].

The heuristic search capability of EA combined with group-based exploration and information exchange among individuals helps to find multiple P-O solutions in a single run, overcoming the limitations of traditional methods.The main goals of MOEA are (1) convergence: to find solutions close to the PF, (2) diversity: to ensure that the solutions are well-distributed, and (3) coverage: to cover the entire PF [17].

Although multi-objective optimisation has the advantage of considering multiple factors at the same time, practical applications may be inefficient if the parameters are not adjusted to meet practical needs. For example, in this study, if the base unit is not set for variations in window size, the algorithm may waste time repeatedly calculating small differences, such as between 1.21 m, 1.22 m and 1.23 m, without adding real value. For example, if the minimum standard size is 1.2 metres, it is irrelevant to calculate whether a 1.3 metre window would provide better light, as in practice it would not be adjusted for such small increments.

1.2 Automatic Generation of Layout Models and Datasets

1.2.1 House-GAN++

Now, many researches have utilized deep learning models to quickly and automatically generate building floor layouts. The fundamental principle of such models is that inputting constraints related to the floor plan and using these inputs to generate corresponding layouts that meet the specified constraints. Taking the representative model House-GAN++ as an example, the following content introduces the basic principles of these models and the issues researchers have been addressing. More importantly, our research highlights a critical problem: the generated building floor plans often lack of window configurations.

House-GAN++ builds on the original House-GAN model(Figure 2), which was first proposed by Nelson Nauata et al. in the paper "House-GAN: Relationally Generative Adversarial Network for Graphically Constrained House Layout Generation."The model structure of House-GAN is shown in Fig. 1, where the generator in the model is based on a Convolutional Message Passing Network (Conv-MPN) designed to generate room layouts based on noise vectors and bubble maps. The input bubble diagram is represented as a graph structure, where each node represents a room, and edges indicate spatial adjacency between rooms. For each room, the model receives a 128-dimensional noise vector and a 10-dimensional room type vector, which are combined into a 138-dimensional feature. This feature is then expanded through a linear layer into an initial 8×8×16 feature tensor. The Conv-MPN module uses the adjacency information in the bubble diagram to perform message passing between neighboring nodes, thereby aggregating and updating room features. Through two rounds of convolutional message passing and upsampling, the feature tensor is expanded to 32×32×16, ensuring that adjacent rooms become more similar in the feature space, ultimately producing a 32×32×1 mask for each room.

During the mask generation process, the message passing mechanism of Conv-MPN effectively exploits the adjacency in the bubble map, thereby enhancing the feature similarity between adjacent room masks. After that, the model enters the testing phase, during which the added thresholding process ensures that adjacent rooms meet the distance constraint, and the boundary detection and post-processing adjustment strategy ensures that the spatial distance between adjacent rooms is within 8 pixels to avoid room overlap. The final room mask is thresholded and fitted into the corresponding orthogonal rectangle to form a complete room layout [18].

16. Gong, M., et al. "EA Applications in Multi-Objective Optimization." Journal of Software 20 (2009): 271-289.

17. Trivedi, A., et al. "MOEA Goals and Applications." IEEE Transactions on Evolutionary Computation 21 (2017): 440-462.

18. Nauata, Nelson & Chang, Kai-Hung & Cheng, Chin-Yi & Mori, Greg & Furukawa, Yasutaka. (2020). House-GAN: Relational Generative Adversarial Networks for Graph-Constrained House Layout Generation. 10.1007/978-3-030-58452-8_10.

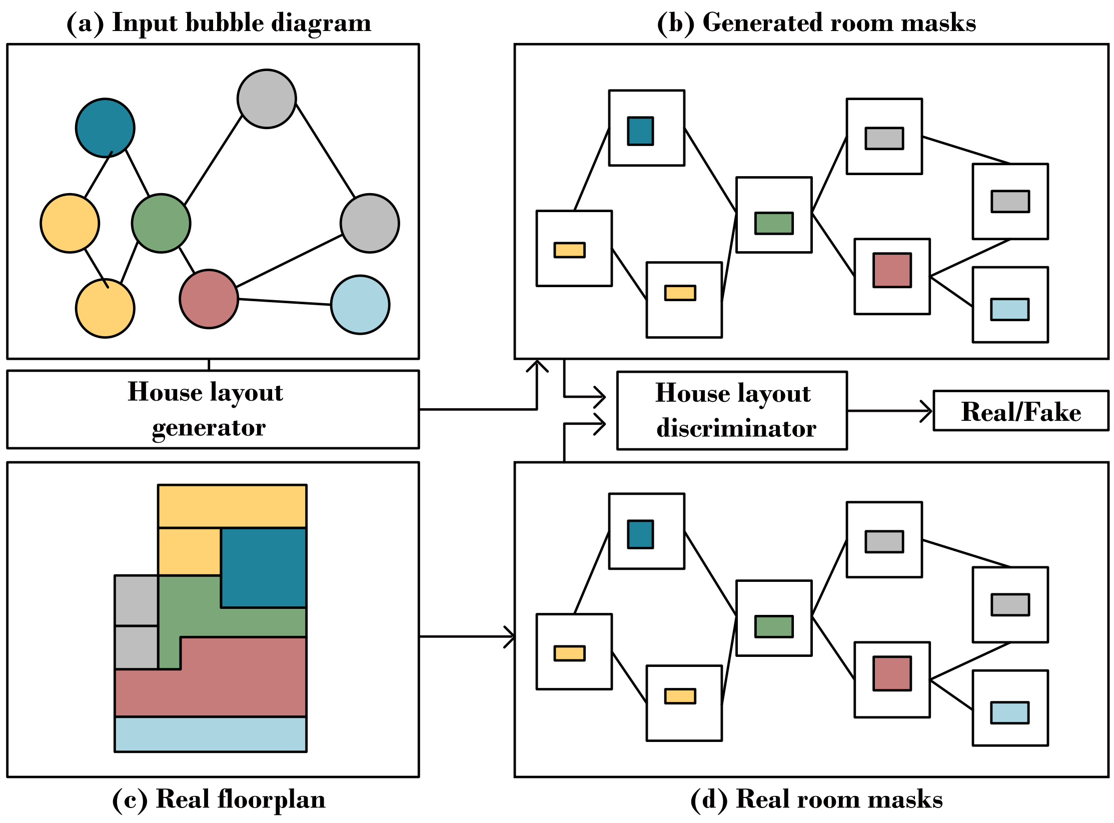


Figure 2[18]

Figure Note: Adapted from "House-GAN: Relational Generative Adversarial Networks for Graph-Constrained House Layout Generation" by Nauata, Nelson, et al., 2020.

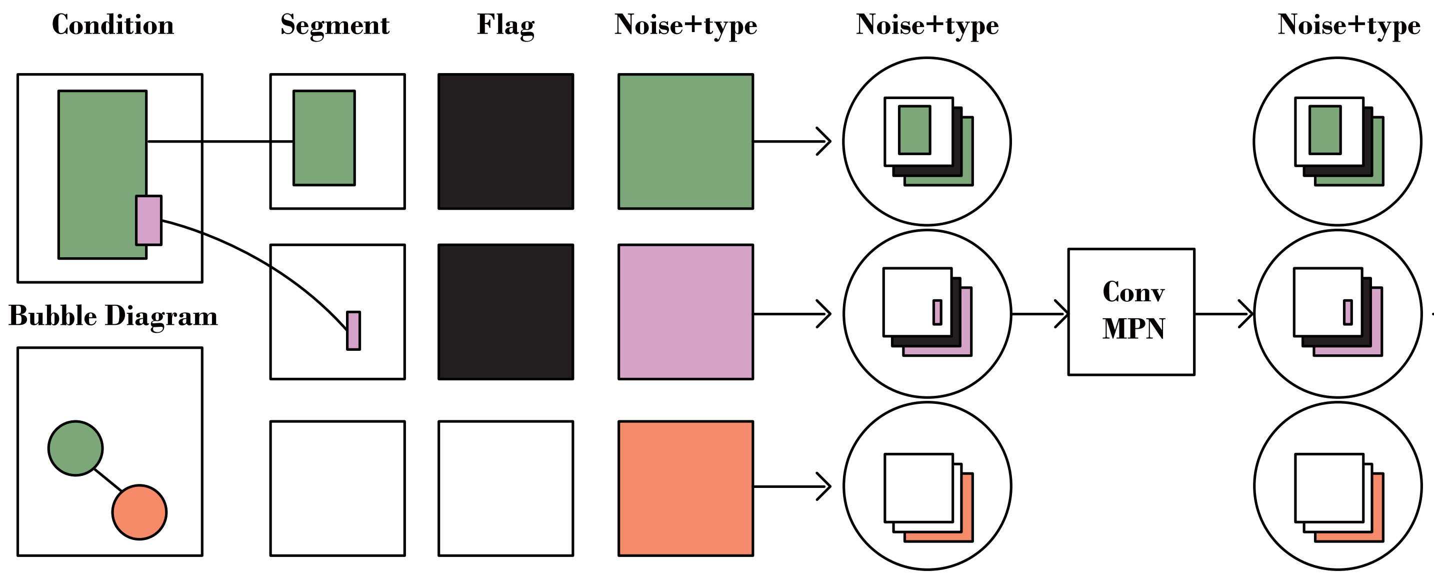


Figure 3 [19]

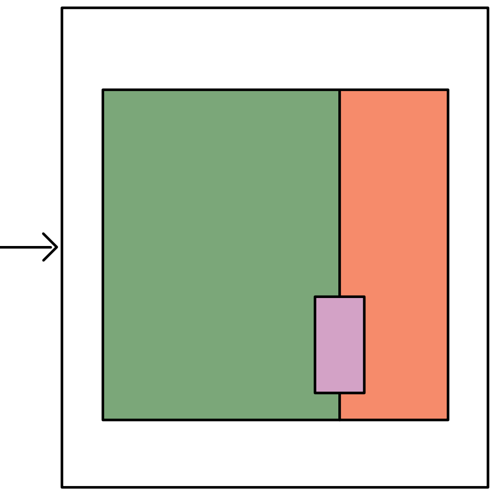
Figure Note: Adapted from "House-GAN++: Generative Adversarial Layout Refinement Network towards Intelligent Computational Agent for Professional Architects" by Nauata, Nelson & Hosseini, Sepidehsadat & Chang, Kai-Hung & Chu, Hang & Cheng, Chin-Yi & Furukawa, Yasutaka., 2021.

However, House-GAN faces challenges when dealing with simpler or more complex input connections. When inputting basic neighborhood relationships, the model may generate layouts with incorrect connections due to the lack of sufficient detail in the input data to accurately constrain the layout. Conversely, if the inputs are very complex adjacencies, the model may miss rooms or misrepresent adjacencies due to a lack of comparable examples in the training dataset. It can also be seen that the room layouts generated by House-GAN do not have door connections between them because there is no feature information for doors in the dataset and no information about doors is encoded.

House-GAN++ is an upgraded version of House-GAN. It addresses these limitations with several key improvements and uses a new dataset called 'RPLAN'.The RPLAN dataset is pre-processed with a series of codes that extract building layout information from images by segmenting image channels and analysing pixel features. This information is stored in the form of a dictionary and includes details such as room types, room and door locations, edges that define rooms, and connections between rooms through doors. In Section 3, we will continue to explain in detail how we extract the required building information from images.

In other parts, House-GAN++ is similar to House-GAN. House-GAN++ is still based on the convolutional message passing network (Conv-MPN) in House-GAN and uses the relationship graph structure of the bubble chart (as shown in Figure 3). This graph structure represents the spatial and functional relationships between rooms. Each node corresponds to a room and each edge represents the adjacent relationship between two spaces.

House layout



19. Nauata, Nelson & Hosseini, Sepidehsadat & Chang, Kai-Hung & Chu, Hang & Cheng, Chin-Yi & Furukawa, Yasutaka. (2021). House-GAN++: Generative Adversarial Layout Refinement Network towards Intelligent Computational Agent for Professional Architects. 13627-13636. 10.1109/CVPR46437.2021.01342.

The model introduces three significant architectural improvements over its predecessor: First, edges, in addition to nodes, carry features related to the generation of doors, which enables more detailed modeling of room connections and architectural elements. Second, each node and edge in the graph is provided with a 2D segmentation mask as an additional input constraint, along with a new loss function that helps guide the generation process towards producing realistic and functional layouts. Lastly, the Conv-MPN feature pooling mechanism is reformulated to allow feature exchange between nodes and edges, enhancing the network's ability to capture complex spatial relationships and improve overall layout coherence.

Together, these improvements lead to a more flexible and accurate model for generating realistic building layouts, resulting in better control of room relationships and architectural constraints during the generation process. It is worth noting that since the edges in the graph not only indicate connections between rooms, but also include information about the shape and location of doors, the building layout generated by House-GAN++ itself contains details about interior doors.

During the generation process, the model takes as input information about the constraints during training, (e.g., the connectivity between rooms and information about the location of the rooms) thus generating a house layout diagram in which each room is represented as an axis-aligned rectangle. In practice, there is an additional step of converting the bubble chart into the corresponding model input information. The bubble diagram is represented as a graph, with nodes signifying rooms and their types, and edges indicating spatial adjacency.

Despite the promising results of House-GAN++ in terms of both accuracy and diversity, its practical use for architects remains limited. Most importantly, these models are mainly used as an early reference tool for generating preliminary building layouts. However, the layouts generated by these models often lack window configurations, which have a significant impact on the building's daylighting, energy efficiency, and ventilation performance. In addition, the lack of window configurations may cause a series of problems in the later design of the building, and may require repeated modifications and adjustments to the building layout in the later stages of the design.

In addition, with sustainability gaining prominence in architecture, residential projects now need to consider factors such as energy efficiency and natural lighting. This presents a challenge: energy and lighting analyses often reveal that better results can be achieved, which requires changes to the layout. These changes require re-running the simulation, which does not guarantee the best results, so adjustments need to be made iteratively based on simulation feedback. In this case, automatic modelling of floor plans does not significantly improve the efficiency of the building design process.

Additionally, the generated layouts typically lack window placements, making it difficult to incorporate environmental factors. In residential buildings, having windows on all sides is impractical, so without windows in the generated layouts, there are no suitable parameters for assessing the design quality in different environmental contexts. However, if the generated layouts include windows optimized for lighting, it becomes possible to evaluate the designs based on the total window area in key rooms. This would allow for more efficient assessments under specific environmental constraints and facilitate regenerating new layouts until the design requirements are met.

1.2.2 RPLAN Dataset

Collecting a sufficient quantity of accurate datasets is a crucial step in training our window size prediction model. In our research, we chose to base our dataset on the building layouts from the House-GAN++ dataset. Using a combination of daylighting simulation software and multi-objective optimization algorithms, we generated window configurations for these layouts. The House-GAN++ dataset itself provides a large variety of diverse building layouts. Furthermore, once our window size prediction program is complete, it can be directly integrated with House-GAN++ or similar automated layout generation models to efficiently post-process the building layouts they produce.

The House-GAN++ paper utilized a dataset called RPLAN, compiled by Wenming Wu et al. The RPLAN dataset was first introduced in their paper, "Data-Driven Interior Plan Generation for Residential Buildings," where they proposed a novel technique for automatically generating floor plans for residential buildings based on a given boundary[20]. This large-scale dataset consists of over 80,000 real residential floor plans, represented as vector graphics with labeled rooms and walls. Each image measures 256x256 pixels, with the green channel storing data about door locations, room outlines and functions, as well as the coordinates and adjacency relationships between different rooms. At the same time, as we have been discussing RPLAN dataset includes information about doors, rooms, and walls but does not provide details on window placements.

20. Wu, Wenming, Xiao-Ming Fu, Rui Tang, Yuhan Wang, Yu-Hao Qi, and Ligang Liu. "Data-Driven Interior Plan Generation for Residential Buildings." ACM Transactions on Graphics 38, no. 6 (2019): Article 234. https://doi.org/10.1145/3355089.3356556.

Part II Related Research References

2.1 Application of Multi-objective Optimisation Algorithms in Architectural and Urban Design Areas

2.1.1 Selection of Multi-objective Optimisation Algorithms

Because of the nature of multi-objective optimization algorithms that can optimize multiple objectives at the same time, it is well suited to the urban design and architectural design process that requires the coordination of multiple external factors at the same time. Therefore, many researchers have tried to combine multi-objective optimisation algorithms with building physics, urban environment and urban form design.

For the use of multi-objective optimisation, we first need to consider the choice of algorithm. Ever since John Holland first proposed the genetic algorithm (GA) in 1975, evolutionary algorithms based on biological simulations have been deeply studied. A GA has many advantages, including parallelism, no need for a derivation or other auxiliary knowledge, the obtainment of multiple solutions concurrently, and an easy implementation. It is considered an effective method for solving a multi-object optimization. The "vector evaluation multi-objective genetic algorithm," originally proposed by Schaffer, is a non-Pareto method[21]. Later, a new algorithm, i.e., the non-dominated sorting genetic algorithm (NSGA), based on the Pareto optimal concept[22], was proposed by Srinivas and Deb. It can search within the feasible region using both parallel and combined methods to find the Pareto solution. Based on previous research, Deb et al. proposed the non-dominated sorting genetic algorithm with the elite strategy (NSGA-II), which takes a step further in searching the efficiency and diversity of the Pareto solution set[23]. Thereafter, in higher-dimensional multi-objective optimization problems, the performance of NSGA-II in the diversity of the solution sets will become dramatically worse. In addition, Deb proposed an improved NSGA-II, called NSGA-III, in 2014[24]. Before the genetic iteration cycle begins, NSGA-III calculates a reference surface, and selects more scattered points for the next generation through the distribution of the solution, which makes the genetic algorithm more robust in solving high dimensional target optimization problems.

For light and energy optimisation in building performance, NSGA-II is sufficient to obtain very accurate results, so even though NSGA-II has been proposed for many years, in recent years multi-objective optimisation algorithms for building performance have mainly used NSGA-II or NSGA-III.

For example, in the paper "A Multi-Objective Optimization Methodology for Window Design Considering Energy Consumption, Thermal Environment, and Visual Performance," Yingni Zhai et al. proposed a similar approach to determine the optimal Window-to-Wall Ratio (WWR). Their study used energy consumption, indoor thermal conditions, and visual comfort as objectives for optimization. By employing the NSGA-II algorithm, they identified the optimal window-to-wall ratio (WWR) by varying the window-to-wall ratio (WWR) of test rooms. These parameters were chosen due to their conflicting relationships; for instance, larger windows can improve visual comfort by allowing more daylight but may also lead to excessive heat gain or loss, affecting thermal conditions and energy use. However, the study only considered daylight illuminance below 500 lux as a target parameter for visual comfort[25].

Qiyan Zhang and colleagues used MOEAs to optimise the layout of urban plots constrained by environmental factors, aiming to improve natural ventilation by balancing architectural and environmental metrics. Their study focuses on two metrics: wind field and frontal area index (FAI). They conducted two experiments: the first optimised the volumetric layout using the NSGA-II algorithm without considering functional configurations, aiming to minimise summer airflow resistance ((λf_summer) and maximise winter airflow ((λf_winter). The second experiment combines the volumetric and functional layouts and uses the NSGA-III algorithm to minimise (λf_summer, maximise (λf_winter and maximise the distribution score ((λd) [26].

In our study, in the multi-objective optimisation section, we specify our objective: to resize the windows in order to change the window-to-wall ratio (WWR) of the wall where the windows are located. By resizing all the windows in the room, we aim to achieve optimal daylight performance. However, in order to effectively implement multi-objective optimisation, we need to further define the performance metrics and design parameters that are of interest during the optimisation process. Therefore, we review recent research on the application of multi-objective optimisation algorithms in architecture.

In his paper entitled 'Finding optimal window-to-wall ratios for office buildings in different European climates and their impact on the total energy saving potential', Francesco Goia provides an in-depth analysis of the impact of various window-to-wall ratios (WWRs) and shading strategies on the energy efficiency of a building in different orientations and climates. His study tested the optimal shading activation fluxes for four building orientations, followed by a paired assessment (north-south and east-west) to determine the optimal window-to-wall ratio (WWR) and the optimal shading strategy. The aim was to determine the optimal design parameters for different building configurations and environmental conditions [27].

Sana Sayadi and co-authors investigated optimal window-to-wall ratio (WWR) configurations across seven different climates using the Köppen-Geiger climate classification. The optimal window-to-wall ratio (WWR) was determined by minimizing the total annual energy consumption (cooling, heating, and lighting). Using standardized test rooms, they evaluated the effects of overhangs and automatic blinds on window-to-wall ratio (WWR) optimization for buildings with integrated automatic lighting control. Additionally, they examined three different window types with various U-values to assess their influence on energy consumption and window-to-wall ratio (WWR). The optimal window-to-wall ratio (WWR) ranges for each climate, orientation, and window type were identified based on these factors[28].

Milad Showkatbakhsh and colleagues propose a methodology for applying Multi-Objective Evolutionary Algorithms (MOEAs) to address urban design complexity caused by conflicting design objectives. The research focuses on balancing design objectives through a multi-objective optimisation algorithm that aims to avoid the subjectivity and rigidity associated with preference-based choices in the design process. This approach consists of three main steps: firstly, defining the design problem; secondly, applying the algorithm; and thirdly, selecting the optimal solution from the generated design alternatives. To this end, the study introduces a data-driven selection mechanism that combines objective and subjective factors to select the optimal solution and tests it in a morphological intervention in urban interstitial spaces. The results show that the framework is not only scalable and adaptable, but also provides effective support to designers in selecting the optimal solution generated by MOEAs, addressing common challenges in applying MOEAs in design [29].

Fatemeh Rezaei and colleagues applied a multi-objective algorithm to enhance thermal and visual comfort in office buildings. They used the percentage of people dissatisfied (PPD) and annual average glare discomfort probability (DGP) as metrics, considering factors such as window-to-wall ratio (WWR) , shading control strategies, viewing angles, glazing transmittance, light shelf length, and height. For each orientation, they determined the maximum and minimum values of PPD and DGP, along with the corresponding decision variables[30].

The above study demonstrates the application of multi-objective optimisation in construction, where a number of performance metrics and design parameters are often used. These cases show that to optimise building parameters using multi-objective optimisation, at least one pair of conflicting objectives must be selected. The algorithm then finds a balanced solution by weighing the conflicting parameters. However, these methods cannot be directly applied to our study. Our dataset consists of real-world building layouts, where the number of windows in each room may not correspond to a single orientation, as tested in the studies described above. Instead, our goal is to simultaneously find the optimal window-to-wall ratio (WWR) for all windows in a room, taking into account different window numbers and orientations. Obviously, it is not feasible to use a single orientation for all windows. In addition, different rooms in a building may not allow windows on all surfaces or only on some parts of the walls. As a result, window-to-wall ratios (WWR) obtained for similar configurations may vary significantly due to differences in total window area. This problem is particularly common in the living rooms in the dataset.

21. J.D. Schaffer, Multiple objective optimization with vector evaluated genetic algorithms, in: Proceedings of the 1st International Conference on Genetic Algorithms, Pittsburgh, PA, USA, July 1985, (1985).

22. N. Srinivas, K. Deb, Muiltiobjective optimization using nondominated sorting in genetic algorithms, Evolut. Comput. 2 (3) (1994) 221–248 ([Online]. Available:), <https://doi.org/10.1162/evco.1994.2.3.221>.

23. K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: nsga-ii, Evolut. Comput. 1917 (2000) 849–858.

24. K. Deb, H. Jain, An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints, IEEE Trans. Evolut. Comput. 18 (4) (2014) 577–601.

25. Zhai, Yingni & Huang, Yanqiu & Meng, Xiaojing. (2018). A multi-objective optimization methodology for window design considering energy consumption, thermal environment and visual performance. Renewable Energy. 134. 10.1016/j.renene.2018.09.024.

26. Zhang, Qiyan, Li, Biao, Li, Hongjian, and Tang, Peng. "Towards Integration and Hybridization in Urban Generation: An Extendable Urban Generative System for Better Natural Ventilation." Proceedings of eCAADe 2023, 2023.

27. Goia, Francesco. "Search for the Optimal Window-to-Wall Ratio in Office Buildings in Different European Climates and the Implications on Total Energy Saving Potential." Solar Energy 132 (2016): 467-492. https://doi.org/10.1016/j.solener.2016.03.031.

28. Sayadi, Sana & Hayati, Abolfazl & Salmanzadeh, Mazyar. (2021). Optimization of Window-to-Wall Ratio for Buildings Located in Different Climates: An IDA-Indoor Climate and Energy Simulation Study. Energies. 14. 1974. 10.3390/en14071974.

29. Showkatbakhsh, Milad, and Mohammed Makki. "Multi-Objective Optimisation of Urban Form: A Framework for Selecting the Optimal Solution." Buildings 12, no. 9 (2022): 1473. https://doi.org/10.3390/buildings12091473.

30. Rezaei, Fatemeh & Sangin, Hamed & Heiranipour, Milad & Attia, Shady. (2024). A Multi-objective Optimization of Window and Light Shelf Design in Office Buildings to Improve Occupants' Thermal and Visual Comfort. Lighting Research & Technology. 11. 55-68. 10.15627/jd.2024.4.

2.2 Predicting Building Performance Using Deep Learning Models

For predictive models, in addition to data collection, it is also very important to choose the right type and structure of predictive models. The type and structure of predictive models have an important impact on the prediction accuracy and efficiency of the model.

We review recent studies on the use of machine learning models to quickly predict building performance results to confirm the model type and structure we will try in our study. In the studies we selected, researchers have demonstrated the effectiveness of machine learning models in predicting building performance, and these models have high prediction accuracy and efficiency.

For example, Yuhao Zhou et al. proposed a meta-modelling workflow for predicting building heat loads in the early design phase. Their model used building characteristics, weather parameters, and operating schedules as inputs to accurately predict hourly cooling and heating loads throughout the year [31]. In their study, the researchers chose as many parameters related to heat load as possible, which requires an in-depth understanding of heat load dynamics. This approach is also very inspiring for our study. By using our own expertise, we can carefully select and pre-process relevant features. This will allow the model to better understand the graphical data it is dealing with and ultimately produce more accurate results.

Similarly, Lei Lei and colleagues proposed a deep learning model for predicting building energy consumption. Their method integrates Entropy Weighted K-means (EWKM, a fast and efficient clustering method for high-dimensional data), Random Forest algorithm, Sparrow Search Algorithm (SSA), and Bi-directional Long and Short-Term Memory Network (BiLSTM) to achieve high accuracy and reliability. They also introduced a feature selection method (EWKM-RF) that combines EWKM with Random Forest to classify and select the factors affecting energy consumption [32].The EWKM-RF method is effective in identifying key factors affecting energy consumption in buildings, which reduces the dimensionality of the data and improves the overall computational efficiency of the prediction model. We draw inspiration from this study in terms of feature processing and loss function design. When dealing with a large number of features, appropriate algorithms can be used to select the most relevant features and ensure that the model focuses on the factors that are most closely related to the results.

Kaoutar Jraida and colleagues used machine learning models to predict the contribution of phase change materials (PCMs) to the reduction of total energy consumption in cities. After comparing various models including ANN, DT, SVM, ELM, GB, RF, TB, GLRM, GPR, LR, GAM, KRRM, and LRR, they found that the SVM model consistently outperformed all other models and was able to successfully predict the hourly network output of buildings [33]. Although our study will not compare so many machine learning models, Kaoutar Jraida's study provides us with an important revelation: when designing a model structure, we can confirm the final model structure by comparing the effectiveness of different machine learning models.

Finally, one thing that needs special attention is that the number of datasets in our study is very different from that in the previous studies. Our dataset is relatively small. Each dataset sample comes from the dataset collection procedure we set up. Although we have significantly improved the collection efficiency by using various methods, the data collection process is still time-consuming. Therefore, it is not feasible to use tens of thousands of samples to train prediction models like the previous studies. In order to break through this limitation, our approach includes comprehensively comparing the performance of multiple prediction models and integrating custom rules with a series of interrelated models. Finally, this method was used to form a high-precision window size prediction procedure. After verification, we found that this method has a huge improvement in accuracy over predictions using a single prediction model.

31. Zhou, Yuhao, Yumin Liang, Yiqun Pan, Xiaolei Yuan, Yurong Xie, and Wenqi Jia. 2022. "A Deep-Learning-Based Meta-Modeling Workflow for Thermal Load Forecasting in Buildings: Method and a Case Study" Buildings 12, no. 2: 177. https://doi.org/10.3390/buildings12020177

32. Lei, Lei & Shao, Suola & Liang, Lixia. (2023). An evolutionary deep learning model based on EWKM, random forest algorithm, SSA and BiLSTM for building energy consumption prediction. Energy. 288. 129795. 10.1016/j.energy.2023.129795.

33. Jraida, Kaoutar & El Mghouchi, Youness & Mourid, Amina & Haidar, Chadia & Alami, Mustapha. (2024). Machine Learning-Based Predicting of PCM-Integrated Building Thermal Performance: An Application Under Various Weather Conditions in Morocco. Journal of Building Engineering. 96. 110395. 10.1016/j.jobe.2024.110395.

Part lll Research Methods

3.1 Preprocessing of RPLAN Datasets

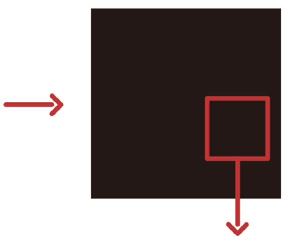
This study differs from the above studies in that our dataset is relatively small. Each dataset sample comes from a combined multi-objective optimisation and illumination simulation process. Although the framework we designed greatly improves efficiency, the data collection process is still time-consuming. Therefore, it is challenging for us to use tens of thousands of samples to train the prediction model, as demonstrated in the previously mentioned studies. To address this limitation, our approach consists of comprehensively comparing the performance of multiple predictive models and integrating custom rules with a range of predictive models. This approach results in a highly accurate window size prediction procedure specifically tailored to make efficient use of the limited available dataset.

In the previous pre-processing method, the preprocessed RPLAN dataset consists of a series of corresponding JSON files[19], containing building information structured in lists labeled "room_type," "boxes," "edges," and "ed_rm."

Convert the png image in the RPLAN dataset to the corresponding json file, mainly through the information of each pixel in the '1' and '2' channels of the image, through the following code to print the image's channel '1 ', print img_room_type to generate a 256x256 two-dimensional matrix, it can be found that each pixel in the room has a specific value to fill, for the gap inside the room, represented by the value '16'. The space outside the plane of the building is filled with the value '13', the value '15' is used to represent the door to the outside of the building, and the

```
img_room_type=img[:,:,1]
print("img_room_type",img_room_type)
```

This is the selection part of the code, img[:,:,1] means to open the image through '1', which is the green channel of the image, and select all the pixels in each row and column of this channel.

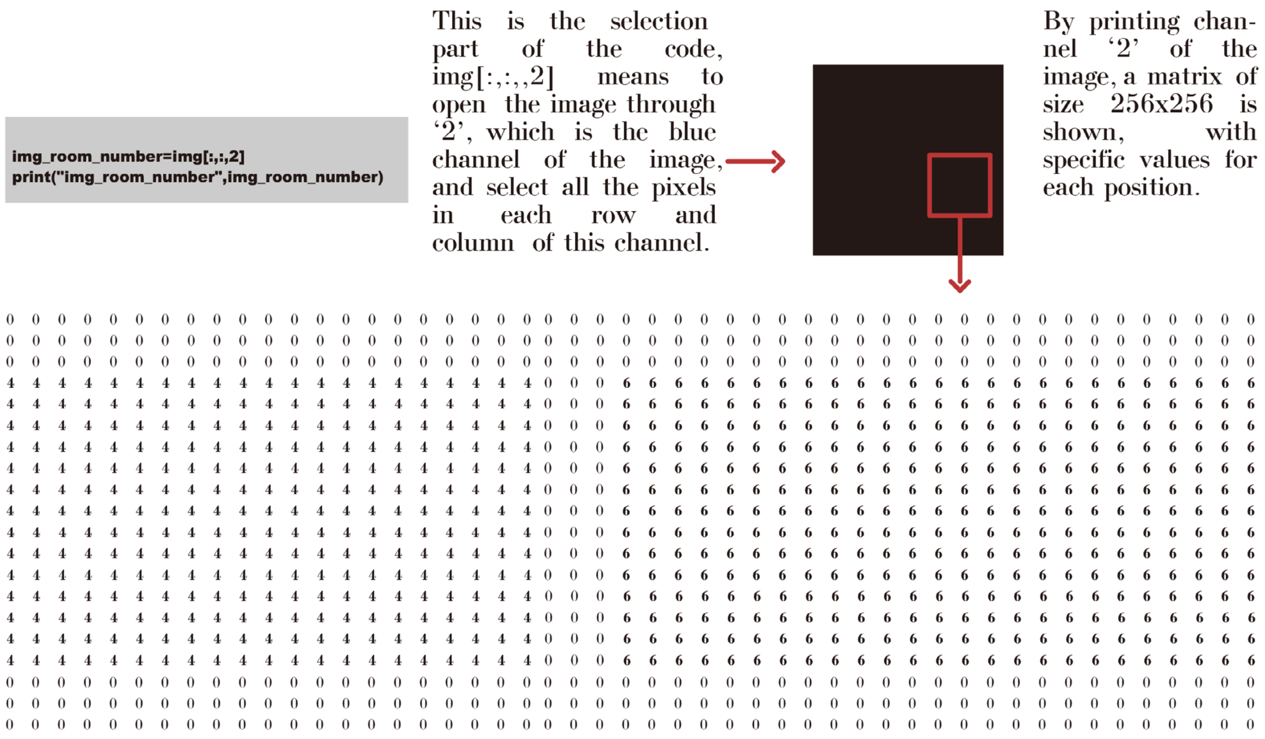


By printing channel '1' of the image, a matrix of size 256x256 is shown, with specific values for each position.

```
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
14 14 15 15 15 15 15 15 15 15 15 15 15 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
14 14 15 15 15 15 15 15 15 15 15 15 15 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
14 14 15 15 15 15 15 15 15 15 15 15 15 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 16 16 16  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 16 16 16  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 17 17 17  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 16 16 16  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 16 16 16  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 16 16 16  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
```

Figure 4. Extracting building plan information from images (drawn by the author)

The function of img_room_number is to encode the pixels belonging to the room locations with the corresponding numbers. The sorting code is unordered, but it fills all the pixels of the room with the number 'n' from the number '1' up to the last room n, so it is possible to know how many rooms there are by the maximum number n in the matrix of img_room_number. Therefore, the maximum number n of the matrix in the img_room_number matrix tells us how many rooms there are(as shown in Figure 5).



```
img_room_number=img[:,:,2]
print("img_room_number",img_room_number)
```

This is the selection part of the code, img[:,:,,2] means to open the image through '2', which is the blue channel of the image, and select all the pixels in each row and column of this channel.
By printing channel '2' of the image, a matrix of size 256x256 is shown, with specific values for each position.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 0 0 0 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 5. Extracting building plan information from images (drawn by the author)

Based on the above information, the rough idea of image preprocessing is:

1. According to img_room_number, there are many rooms in the image, and then each room is put into a 256x256 2-bit matrix, and according to the room pixel position information obtained from img_room_number, the corresponding room pixel value is obtained from img_room_type, and relying on this approach, each room in img_room_type is extracted individually in order to complete the extraction of room_type list information. By this way, each room in img_room_type is extracted individually in turn, and the extraction of room_type list information is completed.

2. After individually extracting each room, internal sector and external sector in img_room_type in turn, set up a function to extract the coordinates of their corners to extract the list of 'boxes' and 'edges', the difference between the two lists is that the former only extracts the diagonal points of the graphic and treats the graphic as a rectangle only, while the latter takes into account all the corner points of the graphic. The difference between these two lists is that the former only extracts the corners of the graph and treats it as a rectangle, while the latter takes all the corners of the graph into account and generates the coordinates of each edge in order.

3. Finally, according to the room, internal department and external department index

information, to generate a list of 'ed_rm' information.

In this format, 'room_type' assigns different numbers to represent the various room types, while 'box' provides the coordinates of the lower left and upper right corners of each room. The "side" list describes the start and end coordinates of each segment, which are associated with the room types on either side. However, rooms can only be connected by internal segments. For example, [[162.0, 59.0, 162.0, 115.0, 3, 0]] indicates that the edge starts at (162.0, 59.0) and ends at (162.0, 115.0). The numbers [3, 0] indicate that this edge is associated with a room of type '3' and has no adjacent rooms of type '0'.

Finally, the 'ed_rm' list represents the indexes of the rooms on either side of each side, corresponding to the 'room_type' index. As mentioned earlier, connections only occur through inner sectors. When there is only one element in the list, it means that the edge is not connected to other edges through the inner sector. For example, [[0], [0, 6], [0], [0]] indicates that all edges connected to the room with index '0' (the first room) are associated with nearby edges. If only one element appears in the list, it means that the specified edge is isolated from other edges by internal sectors(as shown in Figure 6).

{"room_type": [3, 8, 3, 4, 2, 4, 1, 7, 17, 17, 17, 17, 17, 17, 15], "boxes": [[162.0, 59.0, 201.0, 115.0], [55.0, 144.0, 96.0, 197.0], [55.0, 59.0, 95.0, 113.0], [186.0, 119.0, 201.0, 140.0], [162.0, 172.0, 201.0, 197.0], [162.0, 144.0, 201.0, 168.0], [55.0, 59.0, 182.0, 140.0], [100.0, 144.0, 158.0, 197.0], [82.0, 141.0, 94.0, 143.0], [164.0, 116.0, 176.0, 118.0], [80.0, 114.0, 92.0, 116.0], [165.0, 141.0, 173.0, 143.0], [159.0, 175.0, 161.0, 195.0], [105.0, 141.0, 155.0, 143.0], [52.0, 123.0, 54.0, 135.0]], "edges": [[162.0, 59.0, 162.0, 115.0, 3, 0], [162.0, 115.0, 201.0, 115.0, 3, 1], [201.0, 115.0, 201.0, 59.0, 3, 0], [201.0, 59.0, 162.0, 59.0, 3, 0], [55.0, 144.0, 55.0, 197.0, 8, 0], [55.0, 197.0, 96.0, 197.0, 8, 0], [96.0, 197.0, 96.0, 144.0, 8, 0], [96.0, 144.0, 55.0, 144.0, 8, 1], [55.0, 59.0, 55.0, 113.0, 3, 0], [55.0, 113.0, 95.0, 113.0, 3, 1], [95.0, 113.0, 95.0, 59.0, 3, 0], [95.0, 59.0, 55.0, 59.0, 3, 0], [186.0, 119.0, 186.0, 140.0, 4, 0], [186.0, 140.0, 201.0, 140.0, 4, 0], [201.0, 140.0, 201.0, 119.0, 4, 0], [201.0, 119.0, 186.0, 119.0, 4, 0], [162.0, 172.0, 162.0, 197.0, 2, 7], [162.0, 197.0, 201.0, 197.0, 2, 0], [201.0, 197.0, 201.0, 172.0, 2, 0], [201.0, 172.0, 162.0, 172.0, 2, 0], [162.0, 144.0, 162.0, 168.0, 4, 0], [162.0, 168.0, 201.0, 168.0, 4, 0], [201.0, 168.0, 201.0, 144.0, 4, 0], [201.0, 144.0, 162.0, 144.0, 4, 1], [99.0, 59.0, 99.0, 117.0, 1, 0], [99.0, 117.0, 55.0, 117.0, 1, 3], [55.0, 117.0, 55.0, 140.0, 1, 0], [55.0, 140.0, 182.0, 140.0, 1, 7], [182.0, 140.0, 182.0, 119.0, 1, 0], [182.0, 119.0, 158.0, 119.0, 1, 3], [158.0, 119.0, 158.0, 59.0, 1, 0], [158.0, 59.0, 99.0, 59.0, 1, 0], [100.0, 144.0, 100.0, 197.0, 7, 0], [100.0, 197.0, 158.0, 197.0, 7, 0], [158.0, 197.0, 158.0, 144.0, 7, 2], [158.0, 144.0, 100.0, 144.0, 7, 1], [82.0, 141.0, 82.0, 143.0, 17, 0], [82.0, 143.0, 94.0, 143.0, 17, 8], [94.0, 143.0, 94.0, 141.0, 17, 0], [94.0, 141.0, 82.0, 141.0, 17, 1], [164.0, 116.0, 164.0, 118.0, 17, 0], [164.0, 118.0, 176.0, 118.0, 17, 1], [176.0, 118.0, 176.0, 116.0, 17, 0], [176.0, 116.0, 164.0, 116.0, 17, 3], [80.0, 114.0, 80.0, 116.0, 17, 0], [80.0, 116.0, 92.0, 116.0, 17, 1], [92.0, 116.0, 92.0, 114.0, 17, 0], [92.0, 114.0, 80.0, 114.0, 17, 3], [165.0, 141.0, 165.0, 143.0, 17, 0], [165.0, 143.0, 173.0, 143.0, 17, 4], [173.0, 143.0, 173.0, 141.0, 17, 0], [173.0, 141.0, 165.0, 141.0, 17, 1], [159.0, 175.0, 159.0, 195.0, 17, 7], [159.0, 195.0, 161.0, 195.0, 17, 0], [161.0, 195.0, 161.0, 175.0, 17, 2], [161.0, 175.0, 159.0, 175.0, 17, 0], [105.0, 141.0, 105.0, 143.0, 17, 0], [105.0, 143.0, 155.0, 143.0, 17, 7], [155.0, 143.0, 155.0, 141.0, 17, 0], [155.0, 141.0, 105.0, 141.0, 17, 1], [52.0, 123.0, 52.0, 135.0, 15, 0], [52.0, 135.0, 54.0, 135.0, 15, 0], [54.0, 135.0, 54.0, 123.0, 15, 1], [54.0, 123.0, 52.0, 123.0, 15, 0]], "ed_rm": [[0], [0, 6], [0], [0], [1], [1], [1], [1, 6], [2], [2, 6], [2], [2], [3], [3], [3], [3], [4, 7], [4], [4], [4], [5], [5], [5], [5, 6], [6], [6, 2], [6], [6, 7], [6], [6, 0], [6], [6], [7], [7], [7, 4], [7, 6], [8], [8, 1], [8], [8, 6], [9], [9, 6], [9], [9, 0], [10], [10, 6], [10], [10, 2], [11], [11, 5], [11], [11, 6], [12, 7], [12], [12, 4], [12], [13], [13, 7], [13], [13, 6], [14], [14], [14, 6], [14]]}

Figure 6. 14.json display (drawn by the author)

In previous studies, preprocessing the RPLAN dataset was a necessary step for training. However, the information in the raw JSON format is grossly insufficient for our study because it does not include any data about the windows. Furthermore, our experiments require the ability to frequently resize all the windows in the test room. Therefore, we need to add additional data to identify the walls that can accommodate windows in each room. Finally, we also need to add information about the area of these walls in the room, as different sized rooms have different sunlight requirements, which have a significant impact on the size of the windows. This additional information is essential for conducting physical simulations and multi-objective optimisation experiments.

We mainly use the following methods to obtain additional window information data:
1. First, according to the edge length information contained in 'edges', obtain the area information of the corresponding room.
2. Next, we extract the outer contour of the entire building floor plan and capture the coordinates of every pixel position along this outer contour.
2. Next, similar to previous methods, we isolate each room individually. Then, we obtain the contour of each room, gathering the coordinates of every pixel position along these contours.
3. We then match the coordinates of the entire building's outer contour with each room's outer contour. The coordinates that perfectly overlap represent the exterior contour points of each room. Since the line segments in the RPLAN floor plan are straight, and there is a gap between rooms, once we identify the outer contour points of each room, we can extract the start and end coordinates of each line segment. We then calculate the length of each segment and exclude segments shorter than 1.5 meters. The 1.5-meter threshold is determined based on the maximum window-to-wall ratio (0.8) and the standard window unit width (1.2 meters).
4. While identifying the coordinates of each room's outer contour segments, we also retain the information of the room to which each line segment belongs. This includes the index of the room in room_type and the number of outer contour segments within the same room that meet all conditions. At the same time, match the area information already recorded in step 1 with its corresponding outer contour edge, and add the area information to the list where the outer contour edge is located as well.
5. In order to determine the direction of each line segment, we repositioned the two-dimensional matrix for each room based on the coordinates of the start and end points of the outer contours of the rooms. By examining the values in the matrix around the start and end points of the outer contour of each room, we can determine the direction of each line segment. Here, we define the segment directions as follows: the segment located at the bottom edge of the building plan represents the direction towards the south and is labelled '1'; the segment located at the right edge represents the direction towards the east and is labelled '2'; the segment located at the top edge represents the direction towards the north and is labelled "3"; and the segment located at the left edge represents the direction towards the west and is labelled '4'.

In summary, we have expanded the processing of the original dataset by adding new information: the potential number of windows in each room, the coordinates and lengths of walls suitable for window installation, the orientation of these walls, and the corresponding room type and index.

Take the 150.json file in the new dataset as an example:

"ed_windows":

[[145,81,174,81,3,29.0,2,0,1],[225,94,225,132,2,38.0,3,1,2,1906.0],[178,94,225,94,3,47.0,3,1,2,1906.0],[214,136,214,157,2,21.0,4,2,2],[178,157,214,157,1,36.0,4,2,2],[48,108,48,135,4,27.0,3,3,1,1323.0],[31,136,31,175,4,39.0,3,4,4,3153.0],[110,157,110,175,2,18.0,3,4,4,3153.0],[110,157,125,157,1,15.0,3,4,4,3153.0],[31,175,110,175,1,79.0,3,4,4,3153.0],[48,81,48,104,4,23.0,1,5,3,4012.0],[48,81,141,81,3,93.0,1,5,3,4012.0],[143,145,174,145,1,31.0,1,5,3,4012.0]]

The list [225,94,225,132,2,38.0,3,1,2,1906.0] provides details about a line segment representing a window-related wall:

-[225,94] and [225,132] are the coordinates of the start and end points of the segment.

-The number 2 indicates the direction, which in this case is east.

-The value 38.0 represents the length of the wall where the window can be installed.

-The number 3 indicates the room type to which the segment belongs.

-The value 1 represents the room's index in the "room_type" list.

-Then the value 2 indicates that this room has two edges where windows can be installed.

-Finally, the last value 1906.0 indicates the room area in which the edge is located.

3.2 Build a multi-objective optimisation program and Collect New Datasets

The optimal window size, or Window-to-Wall Ratio (WWR), varies depending on specific objectives and the building's environment. In our study, we focused exclusively on daylight as the target for multi-objective optimization, aiming to achieve the best possible indoor lighting conditions by adjusting window sizes. To streamline the process, we simplified our objective to determining the optimal WWR required to achieve the best daylighting in a room.

The reasons for considering only daylight in our optimization are as follows:

1. Time efficiency: Incorporating energy and other objectives into a multi-objective optimisation process can significantly increase computational time. Since our study involves the collection of a large amount of data from both multi-objective optimisation and physical simulation, reducing the computational time for each operation can help to significantly reduce the overall time required for dataset collection.

2. Impact on energy consumption: We refer to the conclusions in "Optimizing Hybrid Ventilation and Daylight-Linked Dimming Control for Carbon Reduction and Thermal Comfort in a Subtropical High-Rise Office Building", which verifies that changes in window size have a relatively small impact on the total energy consumption of the building. [34]

3. The importance of daylight in residential buildings: For residential apartments, light has a more direct impact on the life and work of users.

3.2.1 Visual Building Model in Grasshopper

We chose to create 3D models using the Rhino 3D modeling software because the preprocessed RPLAN dataset generates corresponding JSON files. We needed a program capable of automatically reading these JSON files and converting the data into 3D models. To accomplish this, we used the Grasshopper plugin in Rhino.

In Grasshopper, we mainly use a plugin called 'jSwan'(as shown in Figure 7). This plugin reads JSON files and converts the data into a format that Grasshopper can interpret for model generation. As shown in the figure below, the 'jSwan' plugin reads the information from the JSON file and lists the array of information after the dictionary based on the structure of the JSON file.Then, based on our needs, we use another plugin called "Deserialize Json" to connect to the corresponding dictionary, transforming the dictionary's data into a standard sequence format that Grasshopper can recognize as a list.

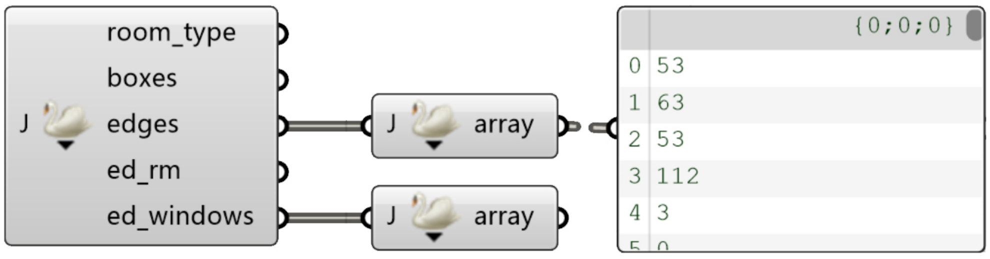


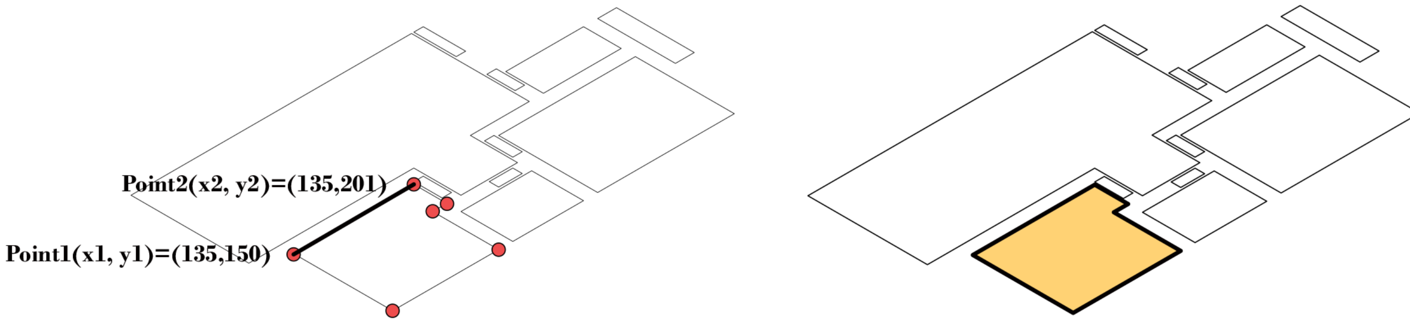
Figure 7. Converting JSON files into Grasshopper-usable data forms (drawn by the author)

34. Yu, Fu Wing & Ho, W.T.. (2023). Optimizing Hybrid Ventilation and Daylight-Linked Dimming Control for Carbon Reduction and Thermal Comfort in a Subtropical High-Rise Office Building. Cleaner Energy Systems. 7. 100096. 10.1016/j.cles.2023.100096.

After converting the data from the JSON files into a format that Grasshopper can use, we utilize basic pl
edges, and ed_windows. The room_type data serves as markers to identify and select specific rooms for tes
"Construct Points" plugin to transform the coordinates from edges into points, which are then connected to
that represent individual rooms. These polylines are then converted into surfaces, and by extruding them u

"edges":[135.0,150.0,135.0,201.0,3,0],[135.0,201.0,177.0,201.0,3,0],[177.0,201.0,177.0,156.0,3,0],[177.0,156.0,149.0,156.0,3,0

Point1 Point2



Conversion of coordinate information in 'edeges' to points, conversion of points to line segments.

Line segment joining to generate closed polylines.

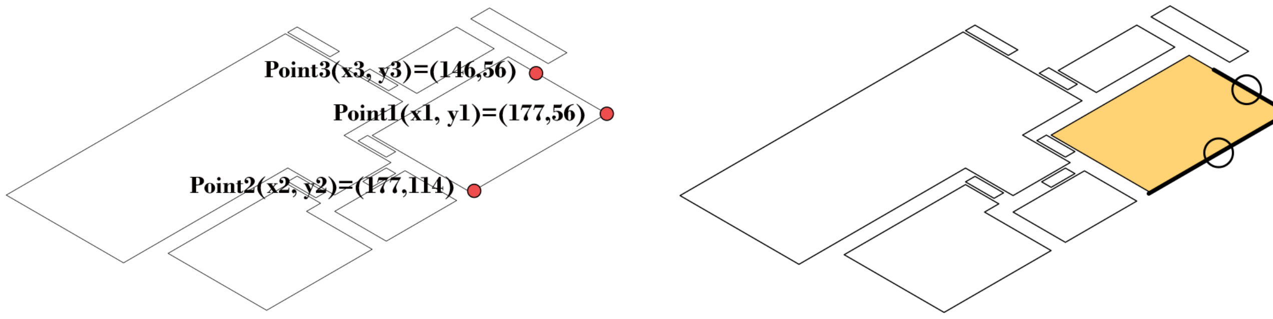
Figure 8. The Coordinates in 'edges' Generate the Building Blocks (drawn by the author)

It is important to note that the 'edge' data contains information about all room boundaries, but we only ne
rooms. To do this, we use the basic components in Grasshopper. The main idea is to first calculate the centr
used to identify overlapping centroids and thus successfully isolate the test room. Finally, the 'Reject Mode'

Similar to the edges, the ed_windows data was collected for the portions of the walls in the floor plan that
the windows varying randomly. In our study, we set the minimum size of the windows to 1.2 metres and adj

"ed_windows":[177,56,177,114,2,58.0,3,2,2],[146,56,177,56,3,31.0,3,2,2]

Point1 Point2 Point3 Point1



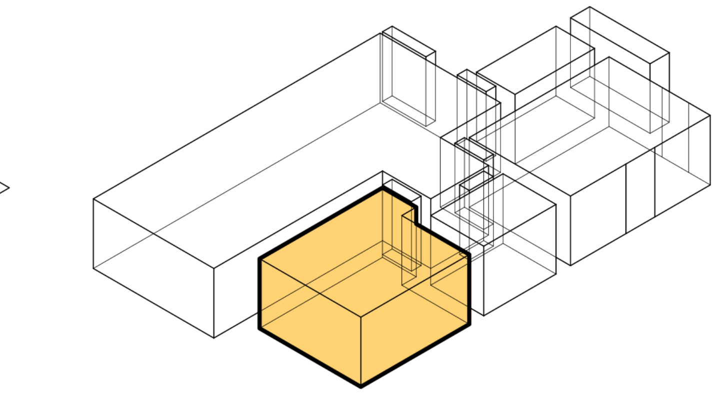
Find the part where the outer contour edges and the test room edges overlap.

Generate the bottom edge of the window.

Figure 9. The Coordinates in 'ed_windows' Generate the Windows (drawn by the author)

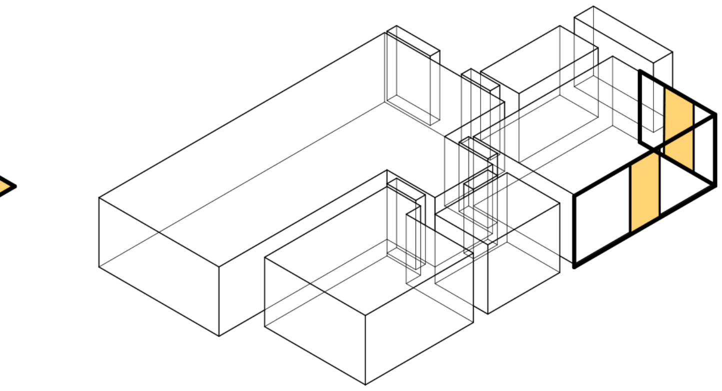
ugins in Grasshopper to construct the corresponding 3D models. The key data we use includes room_type, ting. The edges data is used to construct the mass of each room. The process is as follows: first, we use the form lines. Since each room is separate, the line segments for each room are connected into closed polylines oward, we create enclosed room volumes(as shown in Figure 8).

],[149.0,156.0,149.0,150.0,3,0],[149.0,150.0,135.0,150.0,3,1]


Polyline stretching to generate blocks.

ed specific types of rooms for testing. Therefore, it is important to distinguish the test room from all other point of all rooms (including the test room) using the 'Area' component. Next, the 'Equality' component is component is applied to separate the test room from all other rooms.

were suitable for window placement. Windows were then added in the centre of these walls, with the size of usted them incrementally in 0.6 metre increments(as shown in Figure 9).


Distinguish between a window and a wall by cutting.

It's important to note that rooms such as bathrooms, kitchens, and storage areas do not require an optimal window configuration to achieve the best daylighting conditions. For these types of rooms, window configurations are more focused on functionality. Therefore, these rooms are excluded from the light simulation, and their window configurations are assigned manually based on custom settings.

Specifically, the "Room Type" in the JSON file uses numbers to represent the room type. "1" is the living room, "2" is the kitchen, "3" is the bedroom, "4" is the bathroom, "5" is the balcony, "6" is the corridor, "7" is the dining room, "8" is the study, "10" is the storage room, "15" is the corridor, "16" is the unknown type, and "17" is the internal door.

Different room types have different treatment methods. When optimizing window configuration, rooms such as kitchens, bathrooms, balconies, corridors, dining rooms, storage rooms, corridor doors, unknown types, and internal doors are not considered. Instead, the three room types of living room, bedroom, and study are the focus of research.

3.2.2 Setting of Window Parameters

First, the window height is fixed in this study. There are two main reasons for setting the window height to be the same as the wall height(as shown in Figure 10):

1. Simplify calculation: By assuming that the window height is the same as the building floor height, intermediate calculations can be omitted. This approach eliminates the window height variable, and the room lighting results are directly related to the window width (herein referred to as window size). Therefore, the prediction model only focuses on the relationship between window width and room lighting performance. Although this assumption may introduce some errors, its impact is small and within the acceptable range in residential buildings.

2. Data limitations: The RPLAN data is a planar dataset, which does not contain any information about building heights or window heights. If we set these values manually, researchers who use our data in the future may need to perform a series of conversions before they can use our data. This will reduce the generalizability of our research.

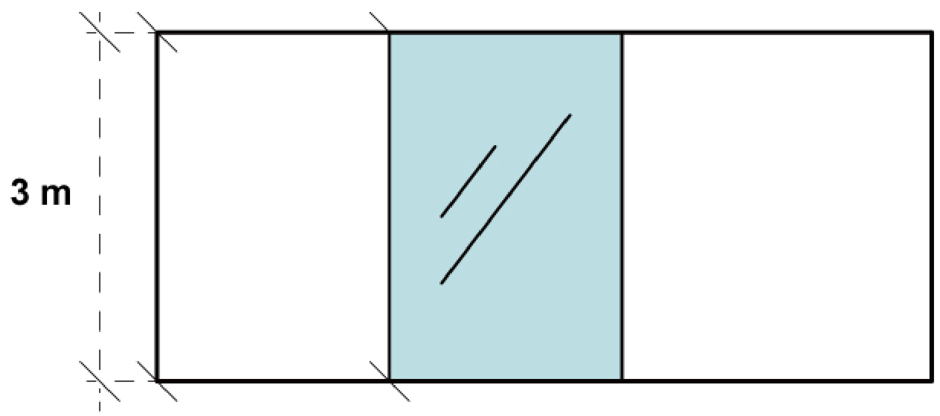


Figure 10. The window height equal to the wall height (drawn by the author)

In the multi-objective optimization process, it is necessary to continuously adjust window sizes to achieve the optimal daylighting results for each room. We proposed a method to adapt the window sizes based on actual requirements. First, the minimum window size was set to 1.2 meters, which aligns with the standard window dimensions in local residential buildings. Next, the minimum scale increment was set to 0.6 meters, corresponding to the size of a typical window frame(as shown in Figure 11).

This approach is implemented using the number slider in Grasshopper, which allows the use of built-in functions. In this component, we entered the function: 1.2 + (x - 1) * 0.6, where n ≥ 0. This function dynamically calculates the final window size, providing flexibility for resizing while maintaining compliance with actual design constraints(as shown in Figure 12).

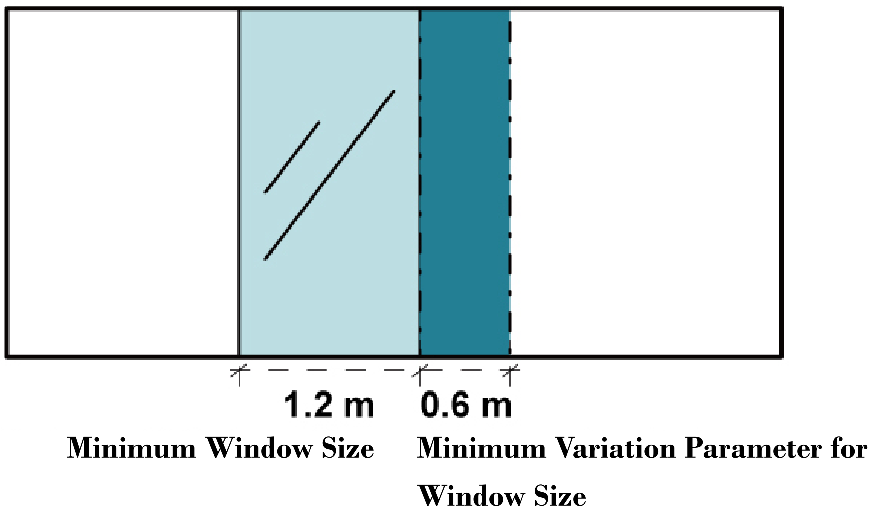
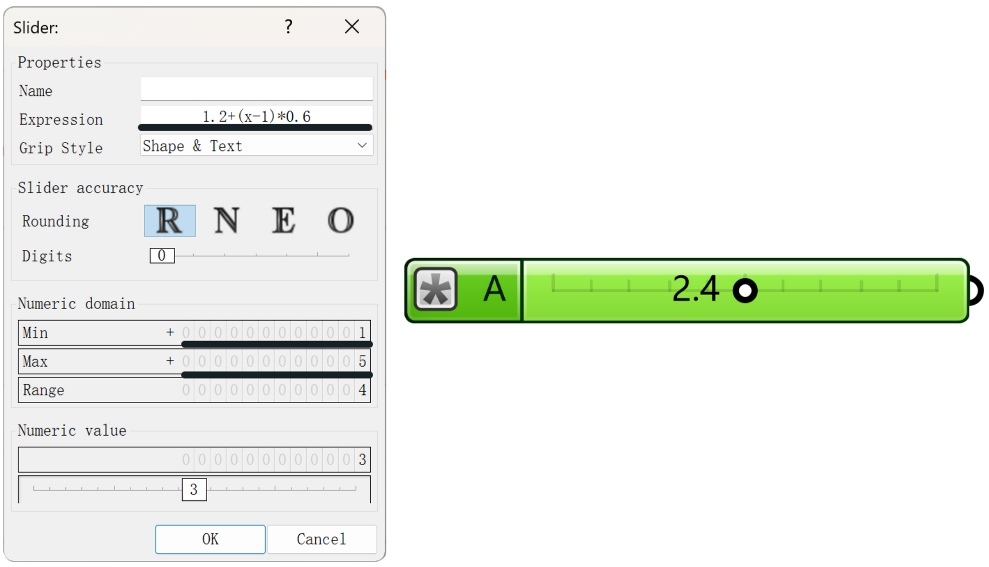


Figure 11. Minimum Size of Windows and Minimum Variation Parameter (drawn by the author)


Slider:

Properties

Name

Expression 1.2+(x-1)*0.6

Grip Style Shape & Text

Slider accuracy

Rounding R N E O

Digits 0

Numeric domain

Min + 0 0 0 0 0 0 0 0 0 0 0 1

Max + 0 0 0 0 0 0 0 0 0 0 0 5

Range 0 0 0 0 0 0 0 0 0 0 0 4

Numeric value

0 0 0 0 0 0 0 0 0 0 0 3

3

Figure 12. Setting Window Parameters (drawn by the author)

3.2.3 Light Simulation

After constructing the building model and configuring the window parameters, we need to connect different parts of the building model to the lighting simulation software. The light simulation process requires the use of a plug-in called "ClimateStudio." To ensure the simulation runs smoothly, the following steps are necessary:

1. Model Setup: Each part of the model needs to be connected to the appropriate input in the ClimateStudio plug-in according to the requirements, and the correct materials must be assigned to different model components. In our research, we simplified certain model components where appropriate. For instance, in the case of windows, we represented the entire window as glass, omitting the window frames. Finally, for the lighting simulation model, the primary elements to be modeled include "surrounding rooms outside the test room," "walls," "ceiling," "floor," and "windows."

Regarding the materials used for glazing(Table 1), walls, ceilings, and floors(Table 2), we selected the most common materials found in residential buildings in Turin, Italy. The materials for the walls, ceilings, and floors are the same, and their properties are detailed in the table below.

The Name of Glazing	Layers	Tvis	Rvis.front	Rvis.back	UVal[W/(m2·K)]	SHGC
Solarban 60 (2) on Starphire -Starphire(Argon)	Double	74.4%	11.0%	12.2%	1.36	0.41

Table 1. The Material Information of Glazing (drawn by the author)

The Name of Glazing	Type	Surface	Roughness	Rvis (tot)	Rvis (diff)	Rvis (spec)	Tvis (tot)	Tvis (diff)	Tvis (spec)
Solarban 60 (2) on Starphire-Starphire(Argon)	Matte	Ceiling	0.00	70.0%	70.0%	70.0%	0.0%	0.0%	0.0%

Table 2. The Material Information of Ceiling, Wall, Ground and material of the surrounding rooms except the Test Room (drawn by the author)

2. Location Selection: The corresponding location for the simulation must be selected to obtain information about coordinates, climate, and radiation intensity.

The next section outlines the climatic conditions at the modelled reference site of Turin.

The daily DBT charts show a gradual increase in the daily temperatures during the hottest months. Hot summers and cold winters with large temperature differences throughout the year. The highest temperatures can be seen in March, April and May, with a minimum of -5.50°C and a maximum of 33.70°C in 2020.

Analysis of HDD and CDD in conjunction with temperature analysis shows that in 2020 there will be a need for heating in June, July and August and heating for most of the rest of the year.

Finally, the schedule(Table 3) is set according to the type of building, which indicates how the users of the space use the space during each hour of the week. This schedule references the classroom schedule settings from the ClimateStudio software, with one key difference: in the classroom schedule, the schedule value for weekends is set to 0. However, in our study, all days for the apartment schedule are set the same. The chart below shows the schedule values for each hour across a 24-hour day, which are consistent across all seven days in our study.

Time	**0-7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20-23**
Schedule Value	0	0.7	1	1	1	0.7	0.5	0.7	1	1	1	0.6	0.2	0

Table 3. Schedule Setting (drawn by the author)

After completing all the settings, a series of operations allow us to obtain various types of light results related to the test room. The 3 main parameters used in our study are the following:

1. Spatial Daylight Autonomy (sDA)

It assesses whether a space receives sufficient daylight on a work plane during standard operating hours on an annual basis. This metric quantifies the fraction of the area within a space for which the daylight autonomy exceeds a specified value.

According to the lES LM-83.12 document, the sDA should be calculated for an illuminance threshold of 300 lx, a DA threshold of 50% and in a time interval from &am to 6pm over the year (sDA300, 50%)(Table 4).However the illuminance threshold and the occupancy period can be changed to better suit to the visual task performed and the type of activity carried out. Similarly, the presence and operation of dynamic blinds should be designed according to the specific requirements.

Classification	**Range**
Preferred	sDA > 75%
Nominally accepted	55% ≤ sDA ≤ 75%
Low	sDA < 55%

Table 4. The preferred value of Spatial Daylight Autonomy (drawn by the author)

2. Annual Sunlight Exposure (ASE)

Annual Sunlight Exposure (ASE): it identifies portions of space receiving too much direct sunlight on an annual basis, which may cause visual discomfort (glare), thermal discomfort or additional cooling costs. This metric quantifies the fraction of the area within a space that exceeds a specified direct sunlight illuminance level for more than a specified number of hours per year over a specified daily schedule with all operable shading devices retracted.

According to the lES LM-83-12 document, the ASE is calculated for a direct sunlight illuminance threshold of 1000 lx (i.e. due to the direct radiation only) and a time threshold of 250 hours(Table 5). In a time interval from 8am to 6pm over the year (ASE1000,250), However, the occupancy period can be changed to better suit the type of activity carried out.

Old classification	Range
Preferred	ASE < 3%
Nominally accepted	3% ≤ ASE < 7%
Low	7% ≤ ASE < 10%
Unacceptable	ASE ≥ 10%

New classification	Range
Exemplary	ASE < 10%
Accepted	10% ≤ ASE < 20%
Unacceptable	ASE ≥ 20%

Table 5. The preferred value of Annual Sunlight Exposure (drawn by the author)

3. Daylight Factor(Mean)

Daylight factor is the most widely used index which defines the percentage ratio of interior illuminance (E) on a horizontal surface to the exterior illuminance (Eh) on a horizontal surface under an overcast[Window to Wall ratio for Day lighting in context of apartment building in Kathmandu valley]. It is also an important parameter to measure the effectiveness of daylight as it expresses the daylight inside any room with respect to external horizontal illumination.

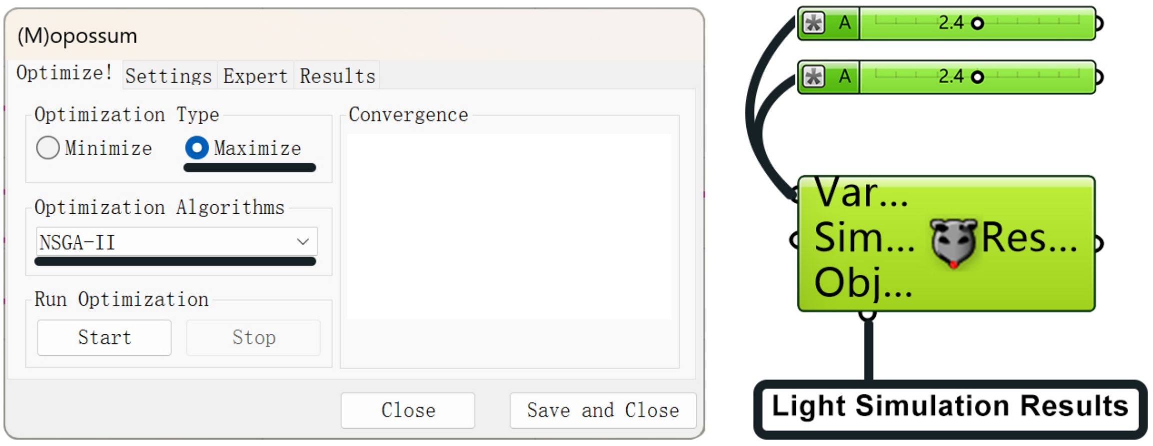
$$DF(\%) = (E_{Outdoor}/E_{Interior})\times 100$$

$E_{Interior}$ is the indoor illuminance (lux) on a horizontal plane. $E_{Outdoor}$ is the outdoor illuminance (lux) on the horizontal plane under cloudy conditions. In the actual simulation, the daylight factor is not involved in the multi-objective optimisation calculation. Instead, it is evaluated after the optimisation process. According to the Turin regulations, a daylight factor of more than 3% is sufficient. In our study, we found that as long as the spatial daylight autonomy (sDA) is met, the insolation coefficient also consistently exceeds the 3% threshold, ranging from 6% to 15%.

3.2.4 Set Up the Automated Multi-objective Optimization Procedure

In order to obtain the optimal daylighting simulation parameters for a test room, the key challenge is to automatically adjust the window sizing parameters after each ClimateStudio calculation. This process must be calculated and refined iteratively using a multi-objective optimisation algorithm until the optimal window size is obtained that meets the daylighting requirements.

For this purpose, we use Opossum, Grasshopper's built-in multi-objective optimisation plug-in, which is able to combine the simulation results from ClimateStudio with a window resizing procedure(as shown in Figure 13). By combining ClimateStudio's iterative calculations with automatic window resizing, the plug-in uses a multi-objective optimisation algorithm to determine the optimal window size.



Optimization Type: Minimize [✓ Maximize]

Optimization Algorithms: NSGA-II

Run Optimization: Start Stop

Figure 13. Automating Multi-Objective Optimization Operations with Opossum (drawn by the author)

Among the available multi-objective optimization algorithms, we chose the NSGA-II algorithm. This selection was based on insights drawn from the related research references discussed in the second section of this paper and tailored to the specific requirements of our study.

The multi-objective optimization aims to achieve the best balance between two solar radiation parameters: Spatial Daylight Autonomy (sDA) and Annual Sunlight Exposure (ASE). It is important to note that in the Grasshopper multi-objective optimization plugin, we can only specify whether we need to maximize or minimize these parameters. As mentioned before, we want the Spatial Daylight Autonomy (sDA) to be as high as possible, while for the Annual Sunlight Exposure(ASE), our goal is to minimize its value. Therefore, we multiply the ASE value by -1, thus converting it to a negative number. By converting the ASE result to a negative number, maximizing its value actually means bringing it closer to zero, thereby minimizing the ASE value.

For efficiency, we set the optimization process to stop in two cases: when the number of iterations exceeds 100, or when no improvement is seen after more than 20 consecutive iterations.

3.2.5 Selection of Input Dataset

We did not randomly sample data from more than 70,000 JSON files for multi-objective optimization, because random sampling can easily miss some building plan types. Our approach is to first select specific room categories. Then list all window configurations and finally collect room data for each window configuration. This strategy avoids the possibility that random sampling may lead to underrepresentation of certain window configurations, resulting in poor prediction accuracy.

We finally extracted 258 rooms from more than 70,000 JSON files for optimization, recorded the results of the optimization, and then created our dataset.

The following are the window configuration combinations considered(Table 6):

Room Type 3&8	
Window number	Categories of window orientations combinations
1	1,2,3,4
2	1&2,1&3,1&4,2&3,2&4,3&4,1&1, 2&2, 3&3,4&4
3	Combinations without regard to window orientation
Note: 1 for South, 2 for East, 3 for North, 4 for West	

Room Type 1	
Window number	Categories of window orientations combinations
1	1,2,3,4
2	1&2,1&3,1&4,2&3,2&4,3&4,1&1, 2&2, 3&3,4&4
3	Combinations without regard to window orientation
4	Combinations without regard to window orientation
Note: 1 for South, 2 for East, 3 for North, 4 for West	

Table 6. All window arrangements (drawn by the author)

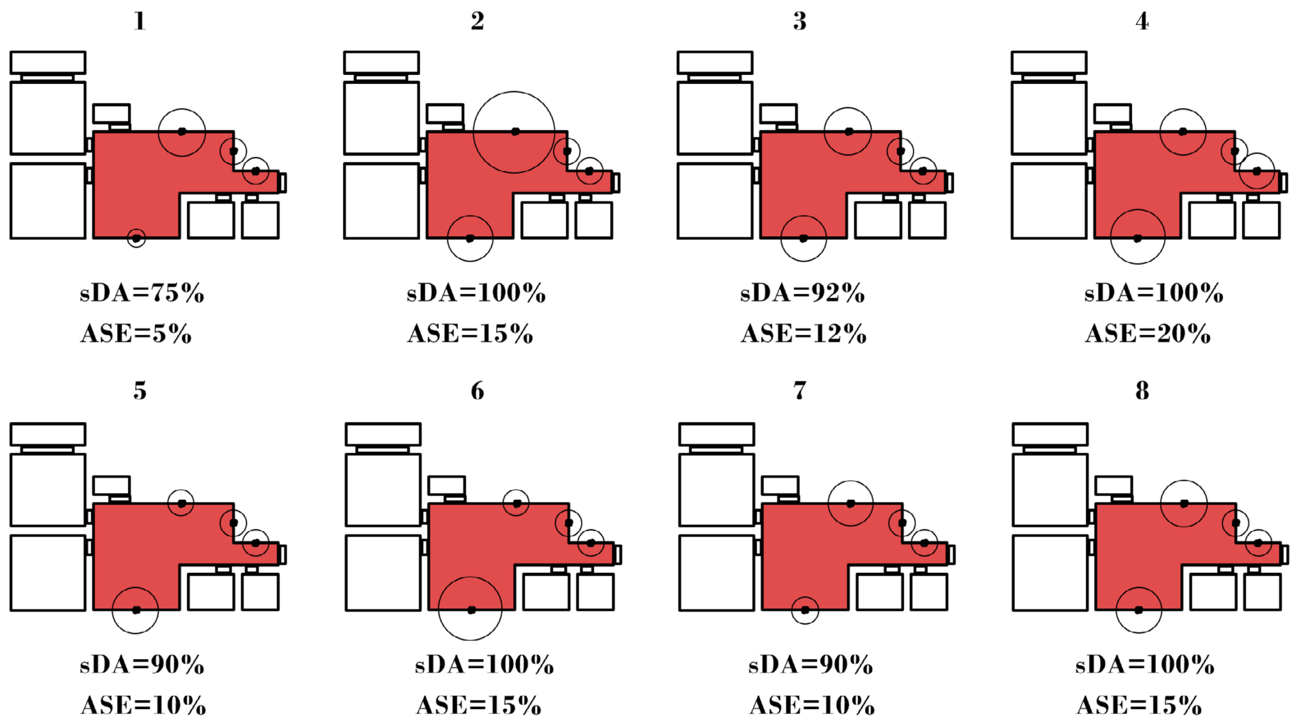
It is worth noting that rooms in categories 3 and 8, representing bedrooms and studies respectively, rarely have four or more windows in both the dataset and the real-life scenarios. Therefore, we excluded cases where these rooms had four or more windows. In addition, when the number of windows reached three or more, the orientation of the windows was no longer considered. This is because a large number of permutations and combinations would result if window orientation was also taken into account. Taking all of these into account would have required a significant amount of additional coding to cover every possibility. Instead, we chose a simple condition of three or more windows and used random sampling to ensure that the various scenarios were well represented in the dataset.

3.2.6 Dataset Collection

3.2.6 Dataset Collection

After completing all configurations, we will run the dataset collection procedure. As we introduced in 3.2.4, the multi-objective optimization procedure will continuously change the size of all windows in the room to obtain the best lighting balance. After the calculation is completed, we will integrate the results with the corresponding graphical output. This includes recording the window and wall sizes for a specific orientation, the relevant window length, window-to-wall ratio (WWR), window orientation, and daylighting indicators. The final result is a comprehensive dataset of window sizes optimized by the multi-objective algorithm to meet the best daylighting requirements.

For each building layout, we record all the edge information of a single room and the final results obtained by the multi-objective optimization process in a structured table(Figure 14).

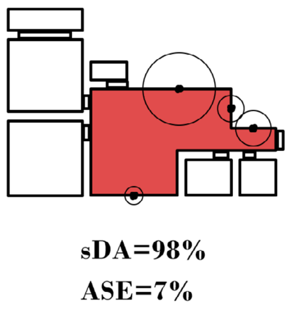


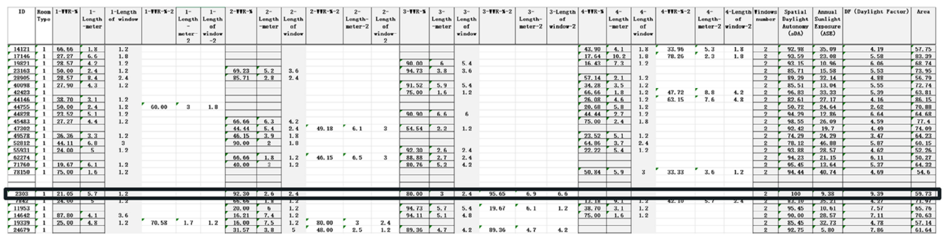
......

Keep updating until more than 20 iterations have not generated a better result

89

Final Result





ID	Room Type	1-WWR-%	1-Length-meter	1-Length of window	1-WWR-%-2	1-Length-meter-2	1-Length of window-2	2-WWR-%	2-Length-meter	2-Length of window	2-WWR-%-2	2-Length-meter-2	2-Length of window-2	3-WWR-%	3-Length-meter	3-Length of window	3-WWR-%-2	3-Length-meter-2	3-Length of window-2	4-WWR-%	4-Length-meter	4-Length of window	4-WWR-%-2	4-Length-meter-2	4-Length of window-2	Windows number	Spatial Daylight Autonomy (sDA)	Annual Sunlight Exposure (ASE)	DF (Daylight Factor)	Area
14121	1	66.66	1.8	1.2																43.90	4.1	1.8	33.96	5.3	1.8	2	92.98	35.09	4.19	57.75
17146	1	27.27	6.6	1.8																17.64	10.2	1.8	78.26	2.3	1.8	2	93.59	23.08	5.58	83.39
19821	1	28.57	4.2	1.2										90.00	6	5.4				16.43	7.3	1.2				2	93.15	10.96	6.06	68.74
23163	1	50.00	2.4	1.2				69.23	5.2	3.6				94.73	3.8	3.6										2	85.71	15.58	5.53	73.95
28905	1	28.57	8.4	2.4				85.71	2.8	2.4										57.14	2.1	1.2				2	89.29	32.14	4.88	56.79
40098	1	27.90	4.3	1.2										91.52	5.9	5.4				34.28	3.5	1.2				2	85.51	13.04	5.55	72.74
42423	1													75.00	1.6	1.2				66.66	1.8	1.2	47.72	8.8	4.2	2	96.83	33.33	5.39	63.81
44146	1	38.70	3.1	1.2																26.08	4.6	1.2	63.15	7.6	4.8	2	82.61	27.17	4.16	86.15
44755	1	50.00	2.4	1.2	60.00	3	1.8													20.68	5.8	1.2				2	50.72	24.64	2.62	70.88
44828	1	23.52	5.1	1.2										90.90	6.6	6				44.44	2.7	1.2				2	94.29	12.86	6.64	64.68
45483	1	27.27	4.4	1.2				66.66	6.3	4.2										75.00	2.4	1.8				2	98.55	26.09	4.59	77.4
47302	1							44.44	5.4	2.4	49.18	6.1	3	54.54	2.2	1.2										2	92.42	19.7	4.49	74.09
49578	1	36.36	3.3	1.2				46.15	3.9	1.8										23.52	5.1	1.2				2	74.29	24.29	3.47	64.23
52812	1	44.11	6.8	3				90.00	2	1.8										64.86	3.7	2.4				2	78.12	46.88	5.87	60.15
55931	1	24.00	5	1.2										92.30	2.6	2.4				22.22	5.4	1.2				2	93.88	28.57	4.62	52.26
62274	1							66.66	1.8	1.2	46.15	6.5	3	88.88	2.7	2.4										2	94.23	21.15	6.11	50.27
71760	1	19.67	6.1	1.2				40.00	3	1.2				80.76	5.2	4.2										2	95.45	13.64	5.37	64.32
78150	1	75.00	1.6	1.2																50.84	5.9	3	33.33	3.6	1.2	2	94.44	40.74	4.69	54.6
2303	1	21.05	5.7	1.2				92.30	2.6	2.4				80.00	3	2.4	95.65	6.9	6.6							2	100	9.38	9.39	59.73
7842	1	24.00	5	1.2				66.66	1.8	1.2										13.18	9.1	1.2	42.10	5.7	2.4	2	83.10	35.21	4.27	71.97
11953	1							20.00	6	1.2				94.73	5.7	5.4	19.67	6.1	1.2	38.70	3.1	1.2				2	95.45	10.61	7.57	65.76
14642	1	87.80	4.1	3.6				16.21	7.4	1.2				94.11	5.1	4.8				75.00	1.6	1.2				2	90.00	28.57	7.11	70.63
19339	1	25.00	4.8	1.2	70.58	1.7	1.2	16.00	7.5	1.2	80.00	3	2.4													2	85.45	32.73	4.78	57.14
24679	1							31.57	3.8	5	48.00	2.5	1.2	89.36	4.7	4.2	89.36	4.7	4.2							2	92.75	5.80	7.86	61.64

Figure 14. Multi-objective Optimization Process and Metrics Logging (drawn by the author)

3.3 Build a Window Size Prediction Program

3.3.1 Data Preprocessing

During the data preprocessing phase, we first select relevant factors influencing window dimensions, integrate this data, and process it into properly shaped feature vectors. In our study, the input features are derived from the reprocessed JSON files of the RPLAN dataset and the data used in the daylight simulation computations.

It is worth noting that we have used data from the ‘ed_windows’ field in the JSON file, but not all of the data has been used. Some of the data (e.g. room types) has been converted to a numerical representation. However, these values are not suitable for direct input into the model as they simply represent the orientation of the edges and lack direct correlation with the predicted window size. For such data, a format conversion is necessary. In our study, the orientation values were encoded using a single encoding. For example, if there are four possible directions, direction ‘1’ is represented as [1, 0, 0, 0], where the position of ‘1’ indicates the direction.

Additionally, not all input features are directly available. For example, to combine the window orientations of a room, we first need to group the edges of the same room in the building layout and then manually construct the features. Since one-hot encoding is used to represent window orientations, the combined feature for a room’s window orientation can be obtained by summing the window orientation vectors within the same room. For instance, if a room has three windows with orientations [1,0,0,0], [0,1,0,0], and [0,1,0,0], the combined window orientation for the room would be [1,2,0,0], where the number "2" indicates that there are two windows facing that direction.

Below are the final input features and their corresponding representations(Table 7).

Input Features	**Forms of Expression**	**Example**
Area	Value	[50]
Room Type	Value	[3]
Direction	One-hot Code	[1,0,0,0]
Windows Number	Value	[3]
Combination of Number of Directions	One-hot Code	[1,2,0,0]
Wall Length	Value	[5.7]
Total Length of Walls	Value	[15.7]

Table 7. Input Features and the Corresponding Representations

3.3.2 Construction of the Program Structure

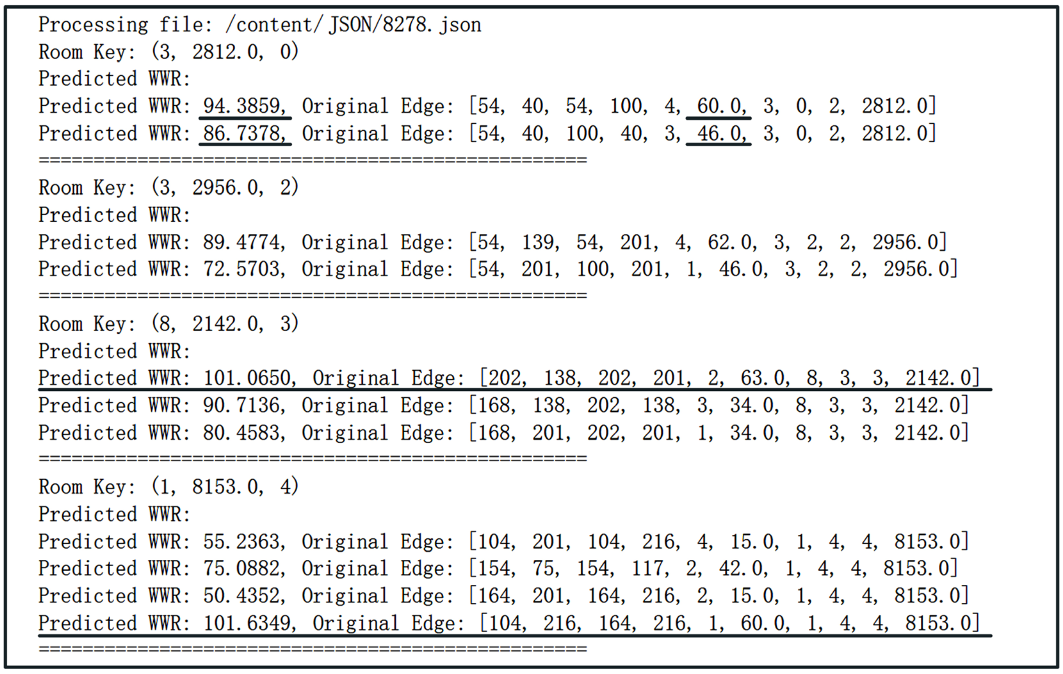
For determining the model structure, we adopted an iterative approach by starting with simple deep learning models and gradually refining the design based on the accuracy of the initial models. We initially experimented with two types of deep learning models:

1. A neural network model with the goal of predicting the continuous regression value, the Window-to-Wall Ratio (WWR).

2. A neural network classification model with the goal of predicting discrete Window Length values.

After training these two individual deep learning models, we randomly selected 10 processed JSON files as test data. The results from both models were unsatisfactory.

The first model, designed to predict the continuous regression value of the Window-to-Wall Ratio, produced particularly poor results. Below is an excerpt of one of the predictions made by the regression model(Figure 15).



```
Processing file: /content/JSON/8278.json
Room Key: (3, 2812.0, 0)
Predicted WWR:
Predicted WWR: 94.3859, Original Edge: [54, 40, 54, 100, 4, 60.0, 3, 0, 2, 2812.0]
Predicted WWR: 86.7378, Original Edge: [54, 40, 100, 40, 3, 46.0, 3, 0, 2, 2812.0]
==================================================
Room Key: (3, 2956.0, 2)
Predicted WWR:
Predicted WWR: 89.4774, Original Edge: [54, 139, 54, 201, 4, 62.0, 3, 2, 2, 2956.0]
Predicted WWR: 72.5703, Original Edge: [54, 201, 100, 201, 1, 46.0, 3, 2, 2, 2956.0]
==================================================
Room Key: (8, 2142.0, 3)
Predicted WWR:
Predicted WWR: 101.0650, Original Edge: [202, 138, 202, 201, 2, 63.0, 8, 3, 3, 2142.0]
Predicted WWR: 90.7136, Original Edge: [168, 138, 202, 138, 3, 34.0, 8, 3, 3, 2142.0]
Predicted WWR: 80.4583, Original Edge: [168, 201, 202, 201, 1, 34.0, 8, 3, 3, 2142.0]
==================================================
Room Key: (1, 8153.0, 4)
Predicted WWR:
Predicted WWR: 55.2363, Original Edge: [104, 201, 104, 216, 4, 15.0, 1, 4, 4, 8153.0]
Predicted WWR: 75.0882, Original Edge: [154, 75, 154, 117, 2, 42.0, 1, 4, 4, 8153.0]
Predicted WWR: 50.4352, Original Edge: [164, 201, 164, 216, 2, 15.0, 1, 4, 4, 8153.0]
Predicted WWR: 101.6349, Original Edge: [104, 216, 164, 216, 1, 60.0, 1, 4, 4, 8153.0]
==================================================
```

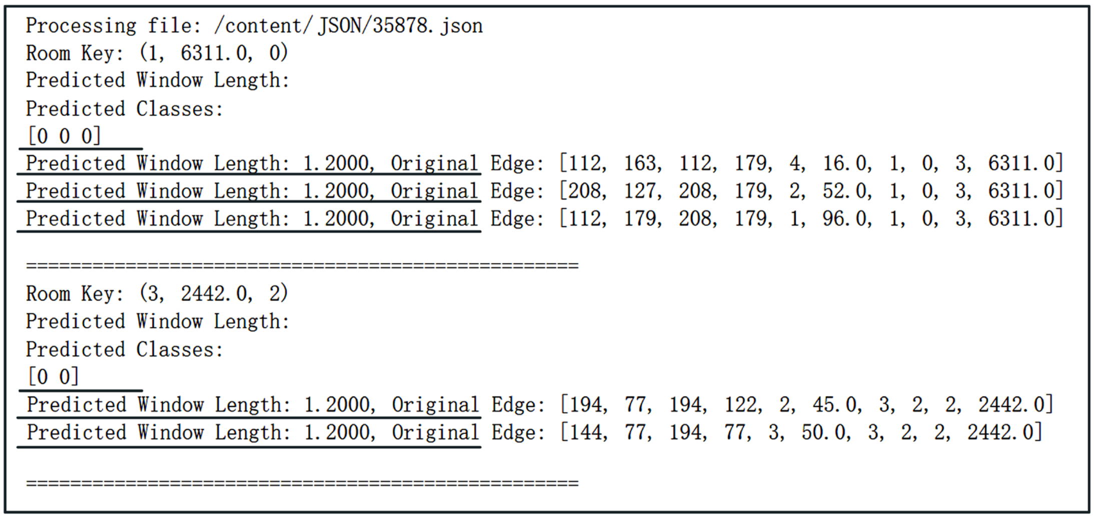
Figure 15. Demonstration of WWR prediction results (drawn by the author)

As can be seen from the results, the predicted window-to-wall ratio (WWR) values are very inaccurate. Firstly, the WWR values are too large and it is clear that WWR has a simple positive correlation with wall length. While this correlation is valid to some extent, in our study this oversimplified relationship resulted in larger WWR values for larger wall lengths - a result that defies basic common sense.

The reasons for these errors can be analysed from three perspectives: the dataset, the prediction target and the model structure. Firstly, in terms of the size of the dataset, we have only 258 room samples. Predicting WWR values from 0 to 100 with such a small dataset will inevitably lead to significant errors.

Secondly, the WWR itself is derived from the ratio of window size to the length of the wall on which it is located. However, since we set constraints such as the minimum window size and the minimum window size increment, the variations in WWR values inherently involve considerable complexity. Finally, regarding the model structure we used, a single neural network prediction model struggles to understand the distribution of windows across different wall orientations. The relationship between the WWR and wall length is also not simply linear. For instance, when a room's daylighting needs are already saturated, even if the length of the south-facing wall increases, to avoid overexposure, the south-facing window size would remain at the minimum of 1.2 meters.

The second model aimed to predict discrete window lengths as classification targets. This classification model categorized window sizes, such as 1.2 meters as category 0, with increments of 0.6 meters forming new categories, until the next category would result in a window size longer than the corresponding wall length. From a design perspective, this approach appears more reasonable because the dataset inherently records window dimensions and uses window sizes as the prediction target, which should theoretically yield more accurate results than predicting WWR. However, testing revealed that this model also produced poor results(Figure16).



```
Processing file: /content/JSON/35878.json
Room Key: (1, 6311.0, 0)
Predicted Window Length:
Predicted Classes:
[0 0 0]
Predicted Window Length: 1.2000, Original Edge: [112, 163, 112, 179, 4, 16.0, 1, 0, 3, 6311.0]
Predicted Window Length: 1.2000, Original Edge: [208, 127, 208, 179, 2, 52.0, 1, 0, 3, 6311.0]
Predicted Window Length: 1.2000, Original Edge: [112, 179, 208, 179, 1, 96.0, 1, 0, 3, 6311.0]

==================================================
Room Key: (3, 2442.0, 2)
Predicted Window Length:
Predicted Classes:
[0 0]
Predicted Window Length: 1.2000, Original Edge: [194, 77, 194, 122, 2, 45.0, 3, 2, 2, 2442.0]
Predicted Window Length: 1.2000, Original Edge: [144, 77, 194, 77, 3, 50.0, 3, 2, 2, 2442.0]

==================================================
```

Figure 16. Window Size Prediction Results (drawn by the author)

The root cause of such results lies in the lack of data and the highly imbalanced distribution of window size categories. By printing the category statistics, we found that in our collected dataset of 258 samples, the distribution of window size categories is as follows(Table 8):

Category	Number
Category 0	270 (48.82%)
Category 1	77 (13.92%)
Category 2	62 (11.21%)
Category 3	33 (5.97%)
Category 4	35 (6.33%)
Category 5	27 (4.88%)
Category 6	21 (3.80%)
Category 7	11 (1.99%)
Category 8	8 (1.45%)
Category 9	2 (0.36%)
Category 10	3 (0.54%)
Category 12	1 (0.18%)
Category 13	3 (0.54%)

Table 8. Category Statistics (drawn by the author)

These results reveal that the distribution of window size categories is highly imbalanced, and most categories have too few samples to form effective classification boundaries. This imbalance explains why our prediction results default to 1.2 meters.

Based on the failure of the previous two attempts, we propose a window size prediction procedure that combines custom rules and multiple models. The procedure consists of the following parts:
1. First, **we trained a model for predicting the total window size in each room (Model 1).** It works by predicting the total window size for the room based on the room and features such as the number of windows and window orientation. This idea comes from the fact that we found a strong correlation in our dataset between rooms and features such as number of windows and window orientation and the total window size for that room. And in the end, the model was validated and found to have good accuracy. It is different from previous classification models that predict window size as a category. The model distinguishes only two categories: a category in which all window sizes are 1.2 metres (the smallest window size), and a category that contains all the remaining categories. The purpose of the classification model is to exclude rooms that already provide sufficient or even excessive sunlight under current conditions and that have the smallest window sizes.

2. Next, **we use the room features, window features and self-attention mechanism to train models that can dynamically analyse and predict the size ratio between windows in a room (Model 2).** The model will calculate the size proportions of different windows in the same room. Then, the total window dimensions calculated by Model 1 are combined to finally obtain the dimensions of each window in the room.

3. Finally, we also introduced **a series of custom rules** to confirm the size of the windows. **The main reason for introducing custom rules is that assigning the dimensions of individual windows in a room based on Model 2 alone sometimes produces large errors that need to be corrected by custom rules.** More than For example, when a window size takes up a large proportion of the wall on which the window is located, the model calculates a window size that is larger than the wall on which it is located. This is shown to be impossible. In addition, the custom rules specify a minimum window size of 1.2 metres and a minimum change in window size of 0.6 metres. This will further correct the window dimensions calculated by Model 2 to produce more accurate results.

In addition, we introduced two loss function mechanisms when training Model 1 and Model 2. A larger penalty is imposed if the predicted window size is less than 1.2 metres. Similarly, another larger penalty is imposed if the predicted window size exceeds the length of the corresponding wall.

The following chart illustrates the structure of the entire program(Figure 17):

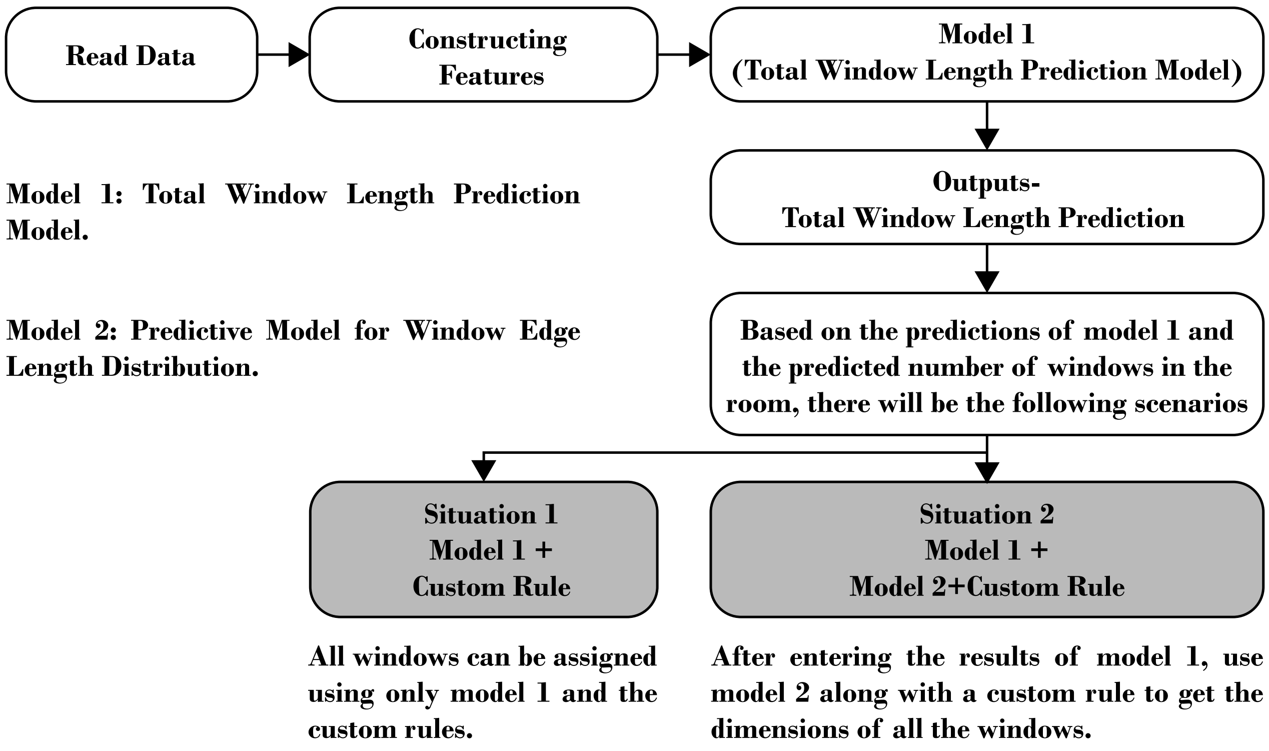

Read Data
Constructing Features
Model 1 (Total Window Length Prediction Model)
Outputs- Total Window Length Prediction
Model 1: Total Window Length Prediction Model.
Model 2: Predictive Model for Window Edge Length Distribution.
Based on the predictions of model 1 and the predicted number of windows in the room, there will be the following scenarios
Situation 1 Model 1 + Custom Rule
Situation 2 Model 1 + Model 2+Custom Rule
All windows can be assigned using only model 1 and the custom rules.
After entering the results of model 1, use model 2 along with a custom rule to get the dimensions of all the windows.

Figure 17. Framework of the Window Size Prediction Program (drawn by the author)

Taking two data as examples, the prediction process of two situations is demonstrated respectively(Figure 18).

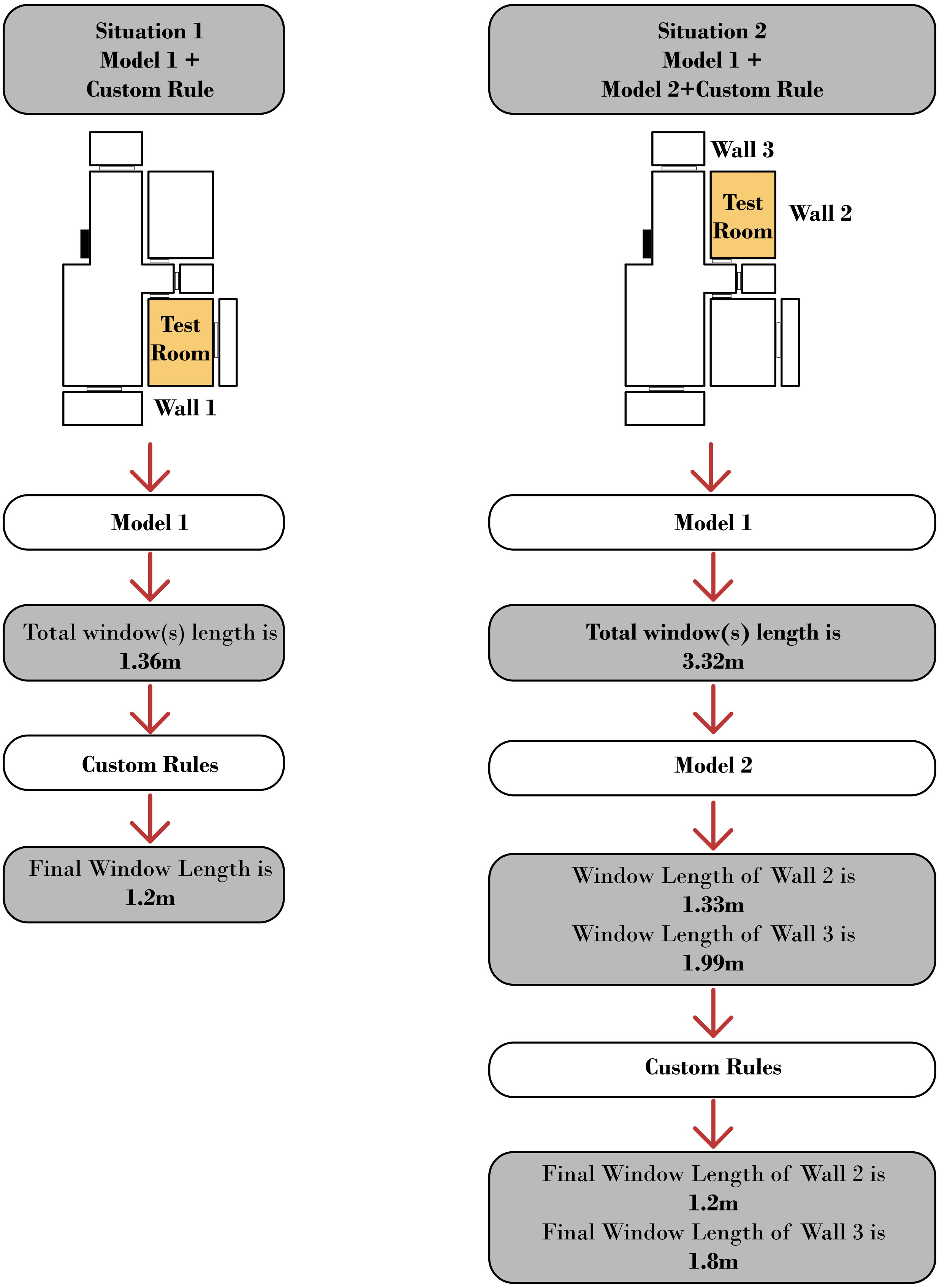


Figure 18. Prediction Program Process Demonstration (drawn by the author)

3.3.3 Validating Model Accuracy

We randomly selected 10 processed JSON files containing 33 rooms and 61 windows as test data. We will check the accuracy of Model 1 and Model 2 separately in the process. There are two main aims. Firstly, we want to improve the accuracy of Model 1 by continuing to fine-tune the model parameters. On the other hand, we want to check if the self-defined rules and Model 2 can assign window sizes accurately and further correct the results of Model 1.

The following are the testing results:

$MSE_{Model\ 1}=2.136$

$MSE_{Model\ 2}=1.163$

The result of $MSE_{Model\ 1}$ is calculated by first calculating the square of the difference between the total window size for each room and the total window size predicted for that room by Model 1, and then averaging those squared differences.

By writing all the $MSE_{Model\ 1}$ results in the table below(Table 9), we found that among the 33 results, there are 2 values that are particularly large. Apart from these, the error rates of the other results are relatively low. We list these two values separately for analysis.

From the tables(Table 10, 11), we can get some simple conclusions. Firstly, the reason for large $MSE_{Model\ 1}$ is that one of the walls with a longer length has a larger prediction error. So first we can know the large prediction error occurs in the walls with a longer length. But at the same time, the walls with a longer length in the room with one or two windows still show high

The Results of $MSE_{Model\ 1}$										
0.04	**0.67**	**0.01**	**0.54**	**0.04**	**22.56**	**17.81**	**1.02**	**1.44**	**0.08**	**1.74**
1.00	**0.19**	**0.03**	**3.24**	**0.92**	**0.42**	**0.00**	**1.25**	**0.00**	**1.30**	**0.67**
0.45	**0.02**	**1.04**	**0.03**	**0.12**	**0.46**	**6.40**	**0.03**	**0.00**	**6.35**	**0.09**

Table 9. The Results of MSEModel 1 (drawn by the author)

The corresponding data for $MSE_{Model\ 1}$ equal to 22.56:

1-Length of Wall	**1-Length of Window**	**1-Length of Window of Prediction**	**3-Length of Wal**	**3-Length of Window**	**3-Length of Window of Prediction**	**4-Length of Wal**	**4-Length of Window**	**4-Length o Window o Predictio**
4	1.2	1.2	7.2	6.6	2.4	4.8	1.8	1.2

Table 10. The Corresponding Data (drawn by the author)

The corresponding data for $MSE_{Model\ 1}$ equal to 17.81:

1-Length of Wall	**1-Length of Window**	**1-Length of Window of Prediction**	**2-Length of Wal**	**2-Length of Window**	**2-Length of Window of Prediction**	**4-Length of Wal**	**4-Length of Window**	**4-Length o Window o Predictio**
5.1	1.2	1.2	11.6	7.8	4.2	2	1.8	1.2

Table 11. The Corresponding Data (drawn by the author)

of f n	**Windows Number**	**Total Window Length**	**Total Window Length of Prediction**
	3	9.6	4.85

of f n	**Windows Number**	**Total Window Length**	**Total Window Length of Prediction**
	3	10.80	6.58

accurary. Based on the the informations, we can draw a basic conclusion that larger errors tend to occur in rooms with more windows and larger walls. However, further research is needed to understand the cause of errors, which will be a follow-up research issue in this direction.

And the $MSE_{Model\ 2}$ is calculated by first computing the squared difference between the predicted and simulated sizes of each window within a room, then summing these values for all windows in the room. Finally, the average of these sums across all rooms is calculated(Table 12).

The Results of $MSE_{Model\ 2}$										
0.00	1.44	0.00	3.60	0.00	18.00	13.32	0.00	1.44	0.08	9.00
6.12	0.36	0.72	3.24	0.36	0.36	0.00	1.44	0.00	0.72	0.36
0.36	0.00	1.44	0.00	0.36	1.44	0.00	0.00	0.00	5.76	0.00

Table 12. The Results of $MSE_{Model\ 2}$ (drawn by the author)

From the table above, it can be observed that rooms with larger errors in $MSE_{Model\ 1}$ also tend to have larger errors in $MSE_{Model\ 2}$, although the errors in $MSE_{Model\ 2}$ are generally lower than those in $MSE_{Model\ 1}$.

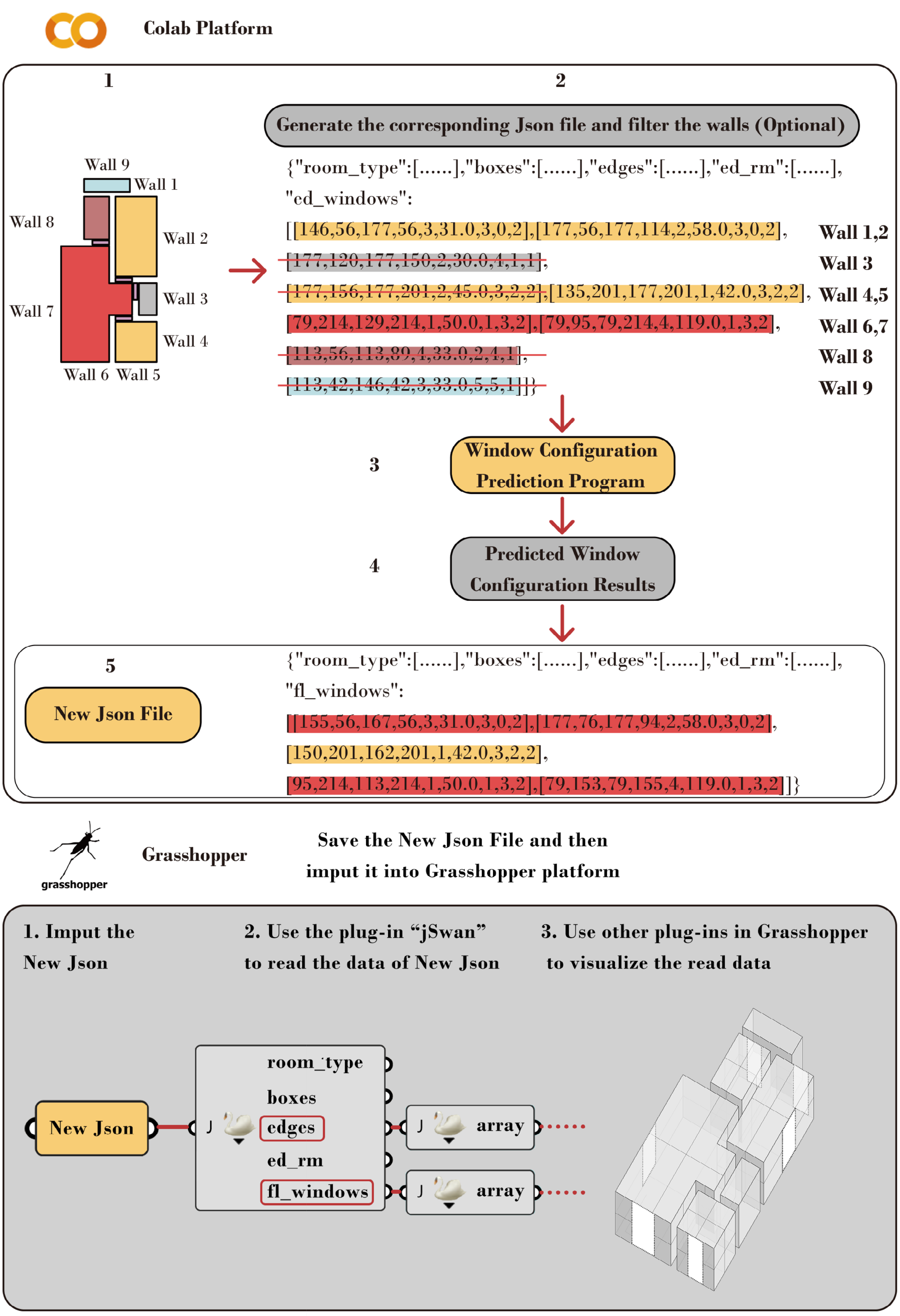
In addition, although some MSE values increased after optimization by Model 2 compared to the results from Model 1, the overall MSE of $MSE_{Model\ 2}$ is still lower than that of $MSE_{Model\ 1}$. This indicates that Model 2, along with the custom rules, contributes to improving the accuracy of the final results.

The data of the rooms whose errors increased after being optimized by Model 2 have been collected for further study. These cases may involve more complex issues, which will not be further elaborated on in this research.

3.3.4 Building an Application Platform

Our application platform consists of Colab platform and Grasshopper platform. First, the floor plan image will be input into the Colab platform and processed into the corresponding Json file. It should be noted that if we want to control the number and direction of the windows in the room later, we need to process the corresponding Json file in this step. Specifically, in the "ed_ windows" list data, delete the data representing the corresponding wall. After that, the new Json will be input into the window prediction program for prediction. After the prediction is completed, the result will be converted into window coordinate data (fl_windows) similar to "ed_ windows", and then a new Json file will be generated.

After obtaining the new Json data, we will import it into the Grasshopper program developed previously, read "edges" and "fl_windows", and generate the corresponding 3D building floor plan block with windows(Figure 19).


Colab Platform
1
2
Generate the corresponding Json file and filter the walls (Optional)
Wall 9
Wall 1
Wall 8
Wall 2
Wall 3
Wall 7
Wall 4
Wall 6 Wall 5
Wall 1,2
Wall 3
Wall 4,5
Wall 6,7
Wall 8
Wall 9
3
Window Configuration Prediction Program
4
Predicted Window Configuration Results
5
New Json File
Grasshopper
grasshopper
Save the New Json File and then impput it into Grasshopper platform
1. Imput the New Json
2. Use the plug-in "jSwan" to read the data of New Json
3. Use other plug-ins in Grasshopper to visualize the read data
New Json
room_type
boxes
edges
ed_rm
fl_windows
array
array

```
{"room_type":[......],"boxes":[......],"edges":[......],"ed_rm":[......],
"ed_windows":
[[146,56,177,56,3,31.0,3,0,2],[177,56,177,114,2,58.0,3,0,2],
~~[177,120,177,150,2,30.0,4,1,1]~~,
~~[177,156,177,201,2,45.0,3,2,2]~~,[135,201,177,201,1,42.0,3,2,2],
[79,214,129,214,1,50.0,1,3,2],[79,95,79,214,4,119.0,1,3,2],
~~[113,56,113,89,4,33.0,2,4,1]~~,
~~[113,42,146,42,3,33.0,5,5,1]~~]}
```

```
{"room_type":[......],"boxes":[......],"edges":[......],"ed_rm":[......],
"fl_windows":
[[155,56,167,56,3,31.0,3,0,2],[177,76,177,94,2,58.0,3,0,2],
[150,201,162,201,1,42.0,3,2,2],
[95,214,113,214,1,50.0,1,3,2],[79,153,79,155,4,119.0,1,3,2]]}
```

Figure 19. The Architecture of the Application Platform(drawn by the author)

Part IV Future Applications, Conclusion and Limitations

4.1 Future Applications

4.1.1 Early Application in Apartment Floor Plan Design

The window size prediction program we developed allows users to input room feature information and obtain the optimal window configuration that meets daylighting requirements for the room(Figure 20). This capability enables designers to rapidly determine the most suitable window configuration for each room during the early stages of apartment floor plan design.

There are two main functions. First, it can immediately observe whether the window size of the building plane meets the requirements. If not, the direction and size of the window opening in the room can be adjusted in advance.

Second, it can control the number of windows. Different users may have different requirements for the number of windows. The prediction program can predict the optimal size of the windows under the corresponding number of windows while controlling the number of windows.

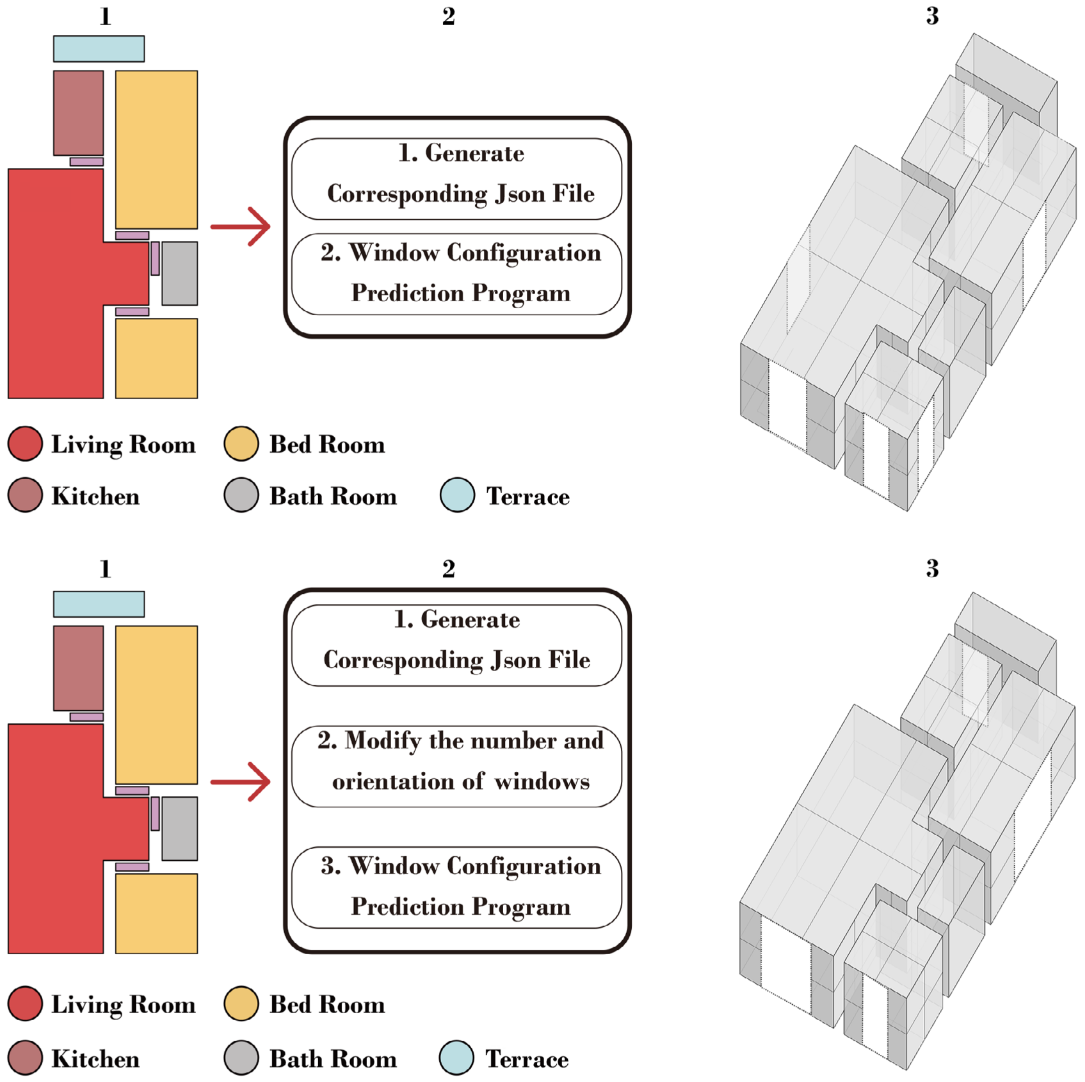

1
2
3
1. Generate Corresponding Json File
2. Window Configuration Prediction Program
Living Room
Bed Room
Kitchen
Bath Room
Terrace
1
2
3
1. Generate Corresponding Json File
2. Modify the number and orientation of windows
3. Window Configuration Prediction Program
Living Room
Bed Room
Kitchen
Bath Room
Terrace

Figure 20. Early Application in Apartment Floor Plan Design (drawn by the author)

4.1.2 Integration with Automated Building Layout Generation Models

The final trained window size predictor program can be integrated with the automatic building layout generation model(Figure 21). Since our dataset is the same as the dataset used by the House-GAN++ model, the predictor is highly compatible with House-GAN++. With this integration, we can parse the building layout generated by House-GAN++, extract the necessary building layout information, and input it into the window size predictor to obtain the window configuration optimized for daylighting performance based on the given layout.

Compared with the previous output, the layout generated by this integration contains the complete window configuration, and at the same time, we can control the number of generated windows by controlling the number of input data, making it more suitable for practical application in floor plan design. Building layouts with optimized window configurations also enable architects to evaluate their daylighting performance at the early stage of design. Subsequent adjustments and optimizations to the layout can take into account both building performance and aesthetic requirements, thereby promoting more informed design decisions.

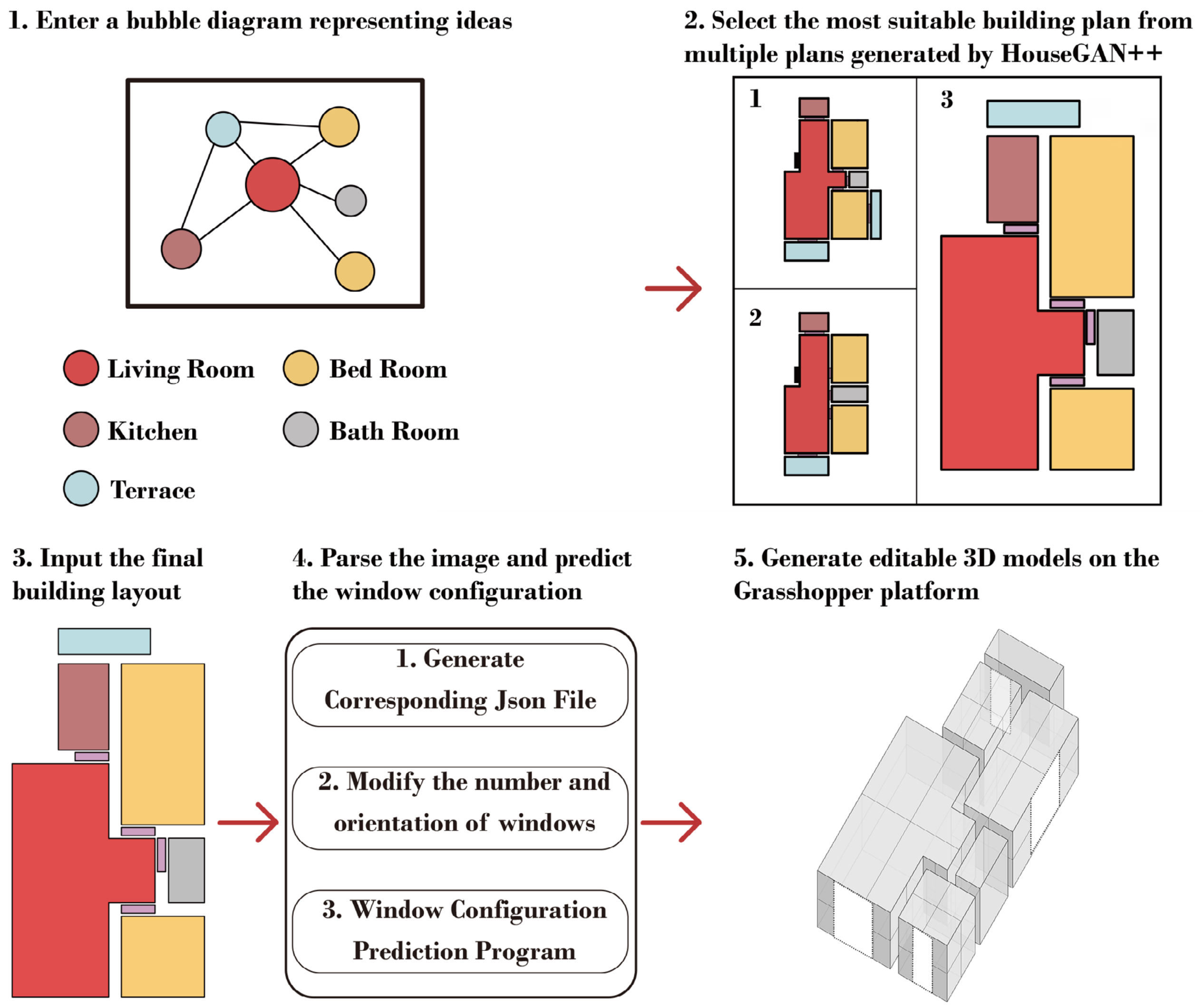

1. Enter a bubble diagram representing ideas
Living Room
Bed Room
Kitchen
Bath Room
Terrace
2. Select the most suitable building plan from multiple plans generated by HouseGAN++
1
3
2
3. Input the final building layout
4. Parse the image and predict the window configuration
1. Generate Corresponding Json File
2. Modify the number and orientation of windows
3. Window Configuration Prediction Program
5. Generate editable 3D models on the Grasshopper platform

Figure 21. Integration with Automated Building Layout Generation Models (drawn by the author)

4.1.3 Combined With Prefabricated Building Design

1. Generate spatial relationships based on user ideas.

2. Generate multiple plan plans and select the most satisfactory one.

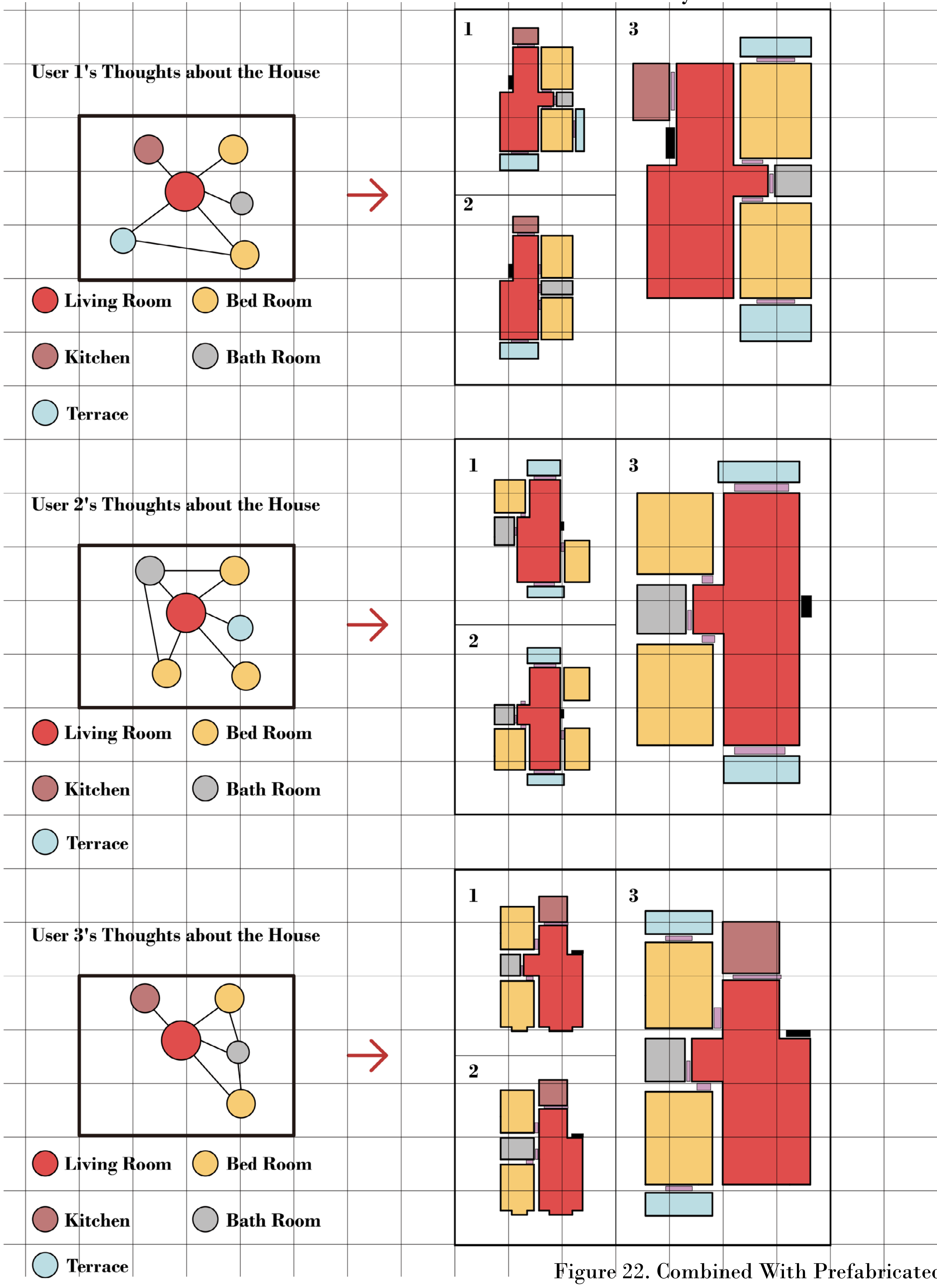
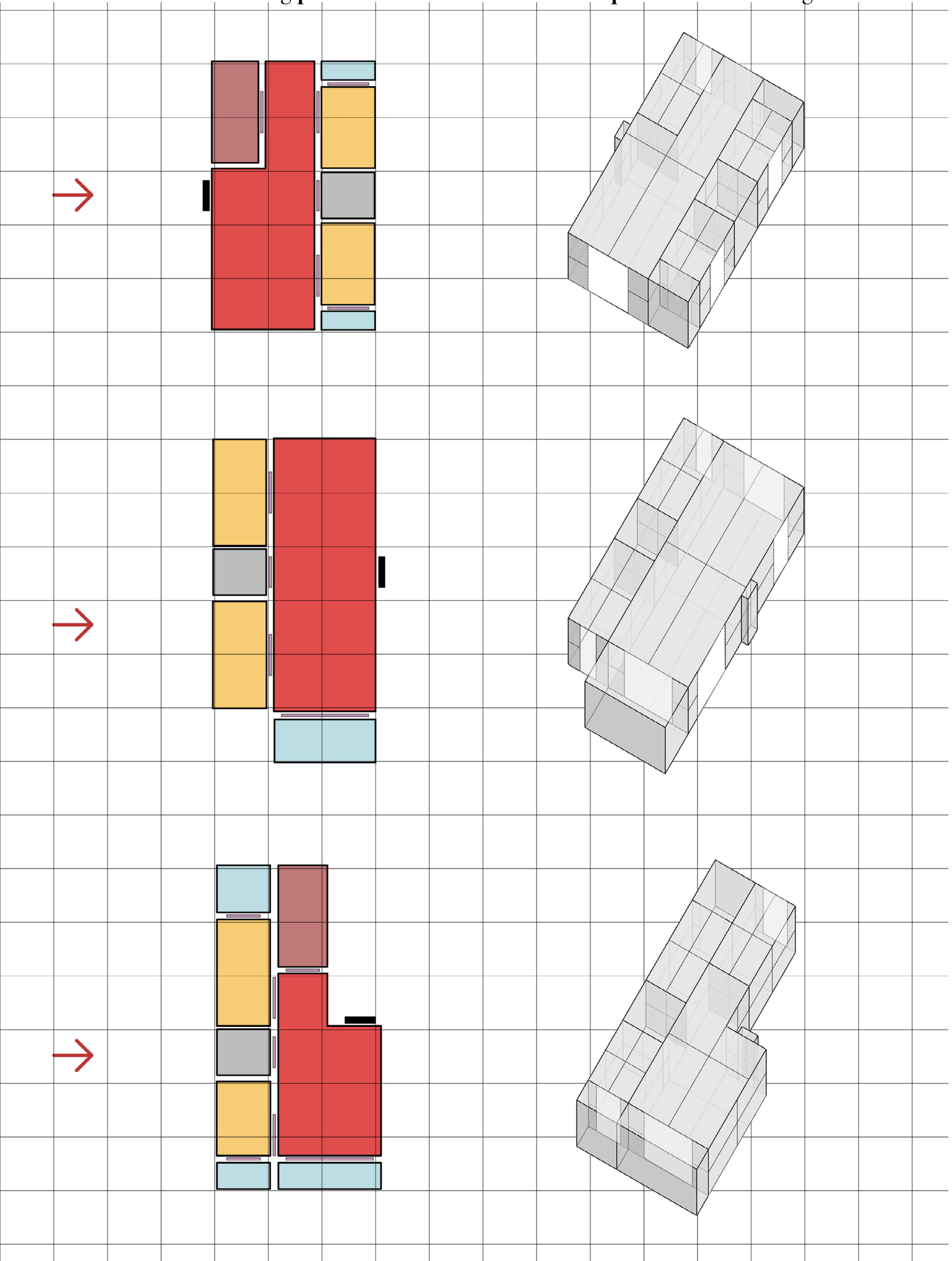


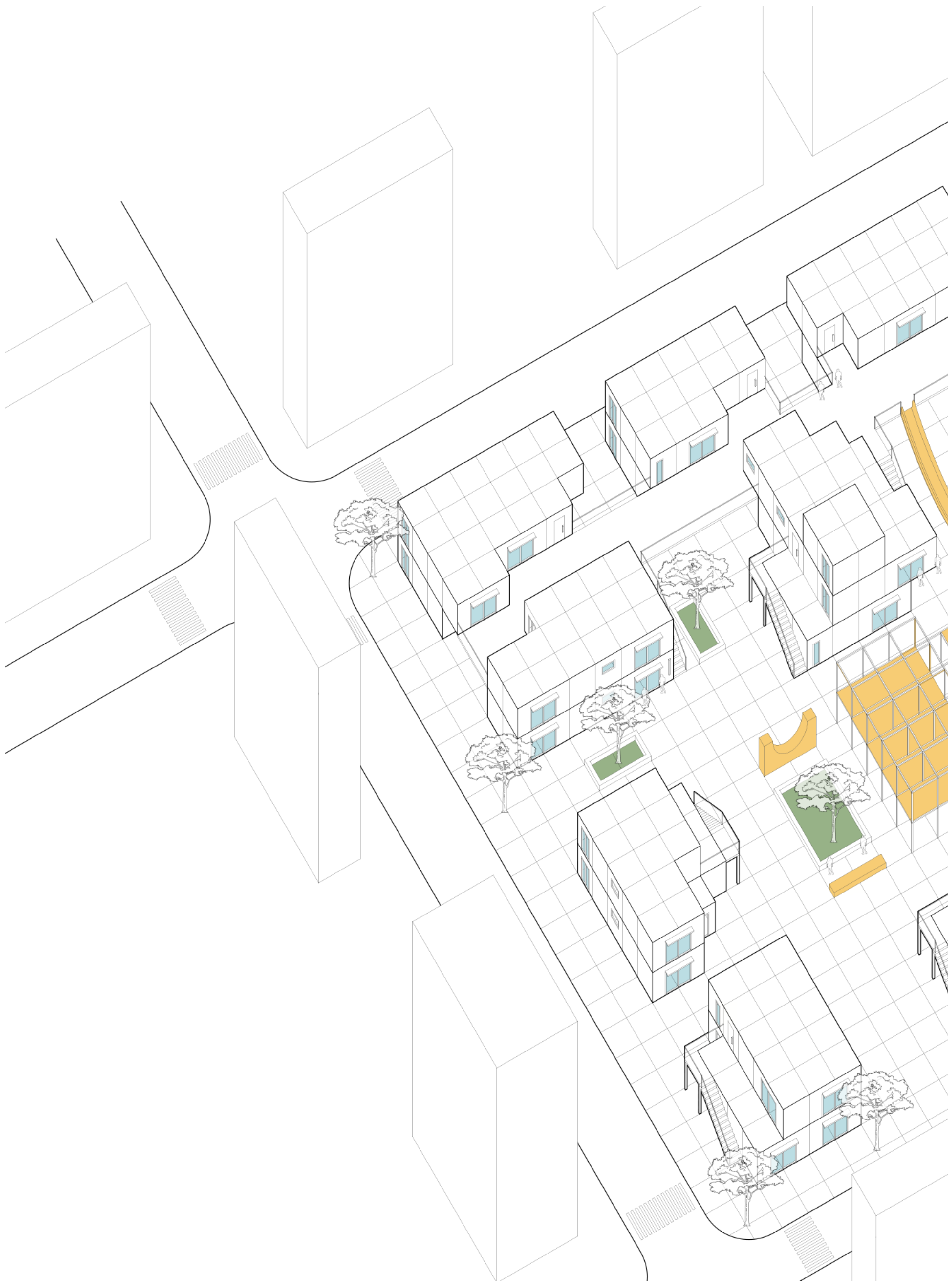
Figure 22. Combined With Prefabricated

3. Modularize the final building plan.

4. Using prediction programs to generate optimal window configurations.



d Building Design (drawn by the author)



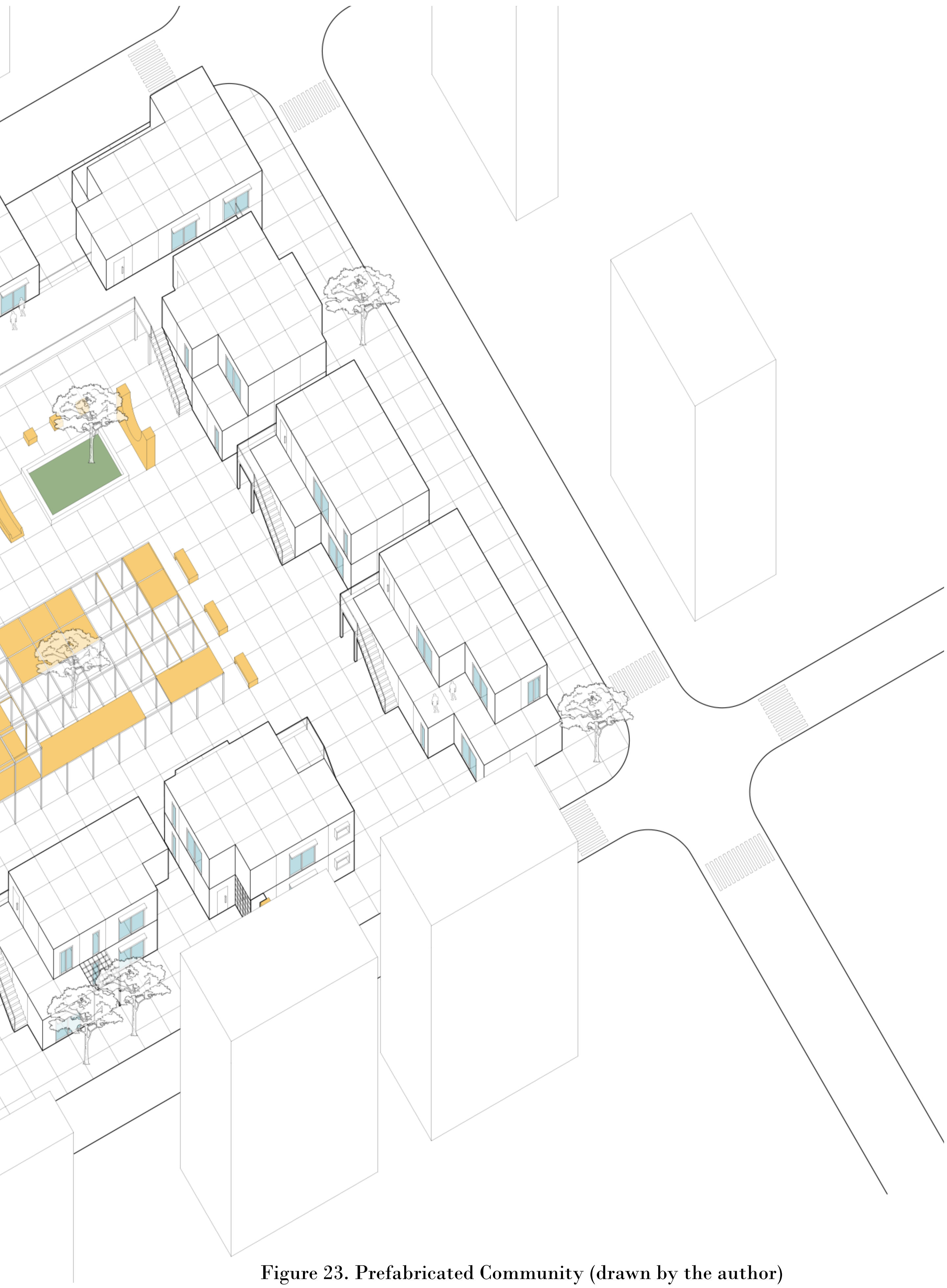


Figure 23. Prefabricated Community (drawn by the author)

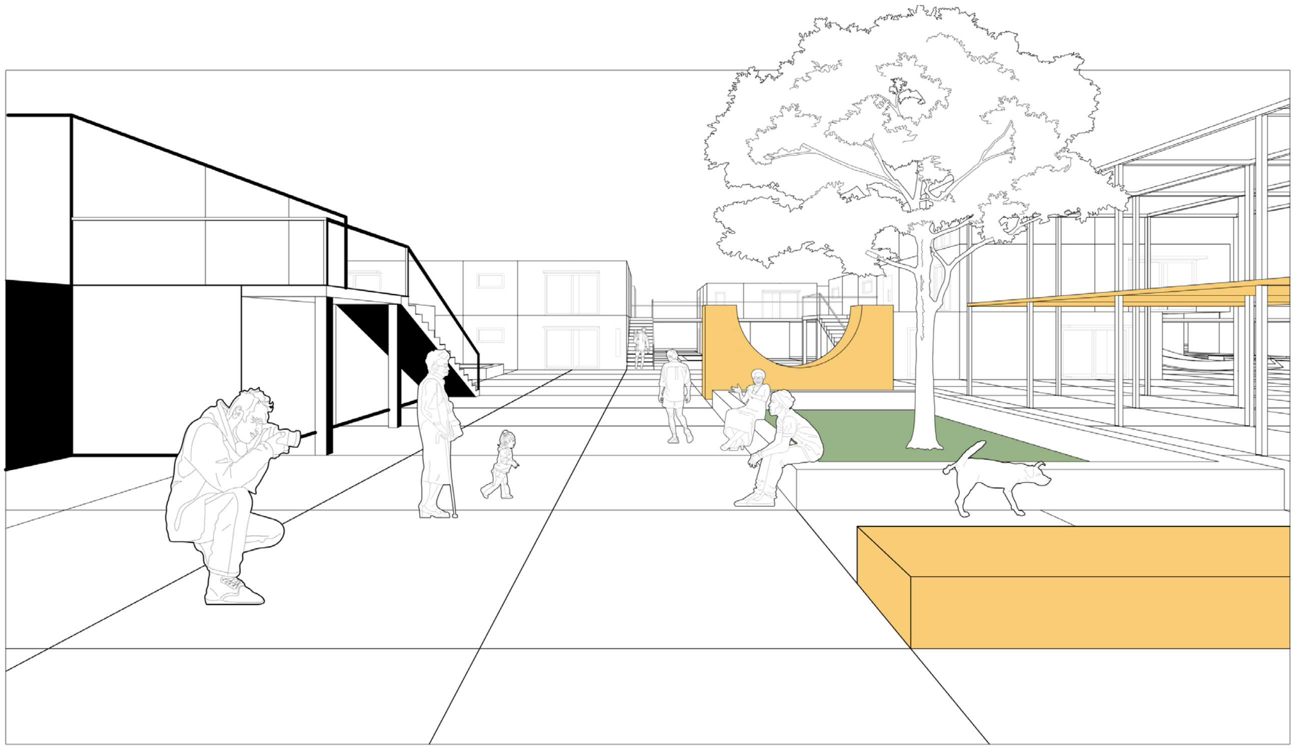


Figure 24. Community Life Scene 1 (drawn by the author)

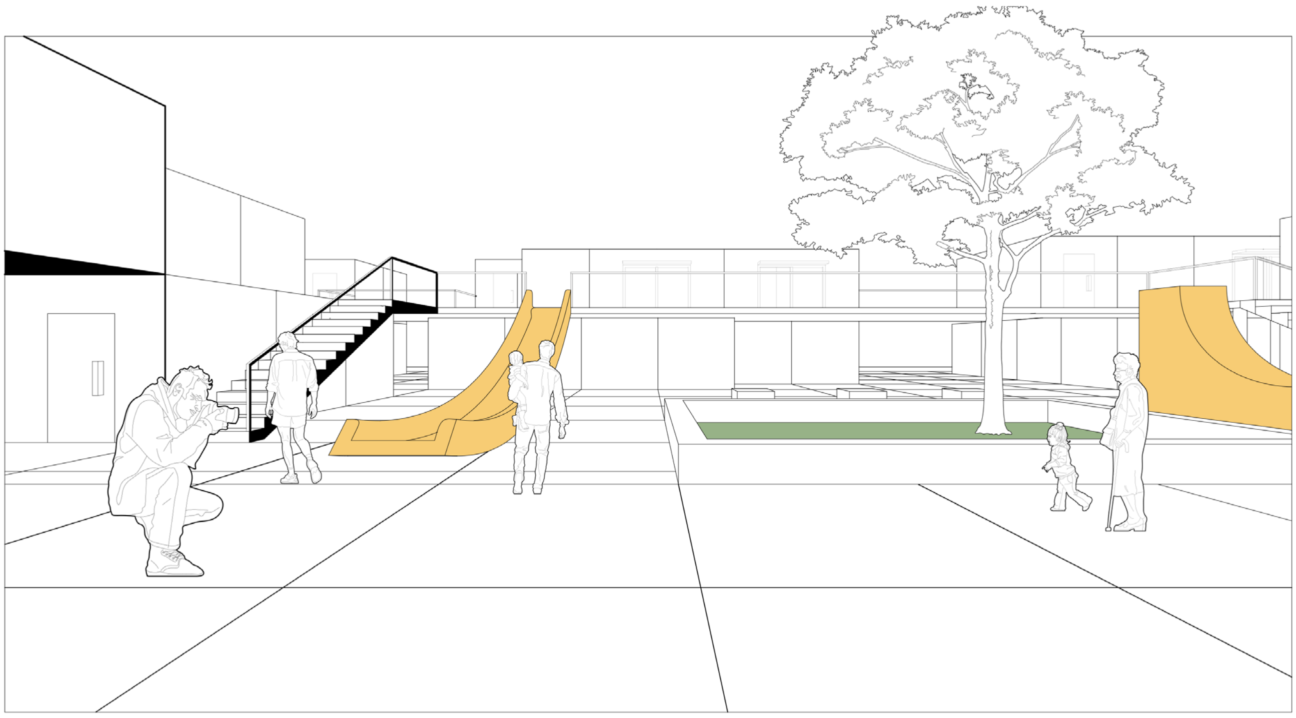


Figure 25. Community Life Scene 2 (drawn by the author)

4.2 Conclusion

In summary, this research tackles a critical gap in automated building layout generation models by addressing the lack of window configurations, which are essential for achieving optimal daylighting performance. This goal is achieved by building a prediction program with multiple models and custom rules combined. It addresses the problem that a single prediction model cannot predict complex building floor plan window configurations with a limited data set.

There are two key innovations in this study, one is the collection of datasets and the other is the prediction procedure built by multiple prediction models and custom rules. In terms of dataset collection, the key innovation lies in converting raw graphical data into textual information, then visualising it in grasshopper software, and finally combining it with light simulation software, multi-objective optimisation algorithms and custom rules to quickly generate accurate window configuration datasets.

In addition, this study highlights the limitations of using a single deep learning model to predict window sizes for small and complex datasets. We significantly improve accuracy by adopting an innovative approach that combines custom rules and multiple prediction models. This approach is computationally efficient while also addressing the challenges posed by unbalanced datasets and complex indoor environments.

Finally, the results of our research can be applied to a variety of fields, including architecture. It can be used to suggest window configurations at the beginning of a building's floor plan design, which can be continuously adapted to the building's floor plan. It can also In addition, by combining it with an artificial intelligence model like HouseGan++, it can quickly provide architects with complete building floor plans and also allow home users to build their own floor plans according to their own intentions. In the smart home sector, it can also be combined with smart furniture such as smart curtains.

4.3 Limitations

Although windows were added to the original building layout and the lighting effect of these windows was verified, the layout itself did not change significantly. In addition, in our study, we only considered the lighting factor and selectively ignored the building physics factors such as energy and ventilation. In addition, for some users, the optimal lighting window configuration may not meet their personal needs. For example, some residents may prefer larger windows in a specific direction for landscape or environmental considerations. Therefore, the optimal window configuration that meets the lighting requirements may not be so important to them.

4.3.1 Limited Physical Building Considerations

As we said before, defining the optimal configuration of windows in a building plan is complex, and it includes factors such as lighting, ventilation, and energy. But in our study, we mainly focus on generating the optimal window configuration in a building layout based on lighting considerations.

But we have shown that we can collect new datasets through multi-objective optimization and use these datasets to train new deep learning models. In future research, we can incorporate more building physics elements into the multi-objective optimization process to obtain new datasets that meet more needs. For example, we can combine energy simulation, wind environment factors, and combine these factors to optimize the functional modules of the building layout.

4.3.2 Regional Limitations of the Program

Since the location of the daylight simulation was chosen to be Turin, Italy, the procedure is subject to regional restrictions. This means that the simulation data is only applicable to areas with similar solar orientation and climatic conditions to Turin. Likewise, the prediction model trained based on this dataset is subject to the same restrictions. Therefore, the collected dataset and the final window size prediction procedure are region-specific and may not be directly applicable to areas with large differences in environmental and climatic conditions.

4.3.3 Daylighting Data in the Collected Dataset Is Based on Ideal Conditions

The prediction program can predict the optimal window size based on optimal lighting balance. However, in the process of collecting data using the lighting software, we found that the ASE value often exceeded 20%, which is an unacceptable threshold. The main reasons for this result are as follows:

1. Maximum window height assumption: The window height is assumed to be equal to the floor height of the building. This results in a larger window area than in the actual situation, which increases the amount of solar radiation entering the room.

2. Ideal surrounding environment: In this study, the test environment for the building layout is assumed to be in an ideal situation with no shading by surrounding buildings. This eliminates the influence of shading by neighboring buildings and further increases the amount of solar radiation.

3. Lack of shading devices: The simulated test room does not have shading devices such as overhangs or curtains. Therefore, the amount of solar radiation is higher than in reality.

Due to these factors, the ASE values in the dataset tend to be overestimated. Therefore, if the predictive model tries to limit the ASE value to determine the optimal window size, the accuracy of the results may decrease and not fully reflect the real-world situation.

ACKNOWLEDGMENTS

I would like to express my special thanks to Professor Lo Turco Massimiliano, Professor Lo Verso Valerio Roberto Maria and Professor Lo Turco Massimiliano's doctoral student Tomalini Andrea. They have given me a lot of help throughout the research process and I am really thankful to them.

Secondly, I would also like to thank my parents and friends who gave me great help while I was completing my thesis.

THANKS

GRAZIE