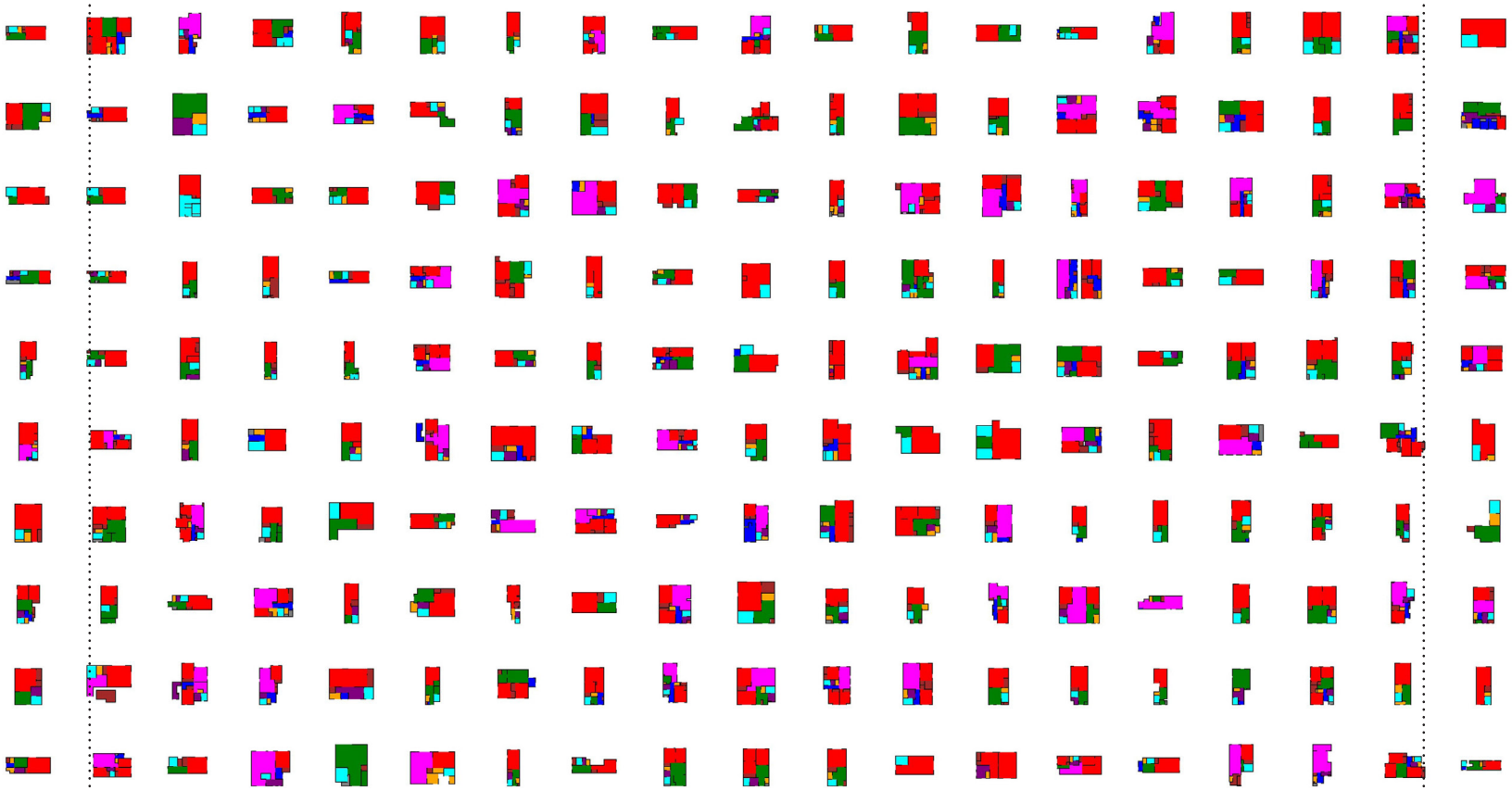
Space Layouts & GANs

GAN-enabled Floor Plan Generation

Stanislas Chaillou, Architect & Data Scientist @ Spacemaker



In this article, we unveil some of our recent results and methodologies implemented at Spacemaker AI's R&D department over the past quarter. This project is one of many ongoing research initiatives, aiming at supporting Spacemaker's long term vision.

Apartment layout is a challenging yet fundamental task for any architect. Knowing how to place rooms, decide their size, find the relevant adjacencies among them, while defining relevant typologies are key concerns that any drafter takes into account while designing floor plans. In this article, **we propose showcasing possibilities offered by Generative Adversarial Neural Networks models (GANs), and their ability to generate relevant floor plan designs**. In short, we turn to GAN models, and more specifically Pix2Pix, to help us design housing floor plans, given a set of initial conditions & constraints.

I. AI & GANs

Generative Adversarial Neural Networks have recently provided evidence of the creative ability of AI models. As any machine-learning model, GANs learn statistically significant phenomena among data presented to them. Their structure, however, represents a breakthrough: made of two key models, the *Generator* and the *Discriminator*, GANs leverage a feedback loop between both models to refine their ability to generate relevant images. The Discriminator is trained to recognize images from a set of data. Properly trained, this model is able to distinguish between a real example, taken out of the dataset, from a "*fake*" image, foreign to the dataset. The Generator, however, is trained to create images resembling images from the same dataset. As the Generator creates images, the Discriminator provides it with feedback. In response, the Generator adapts, to produce even more realistic images. **Through this feedback loop, a GAN slowly builds its ability to create relevant synthetic images**, factoring in phenomena found among observed data.

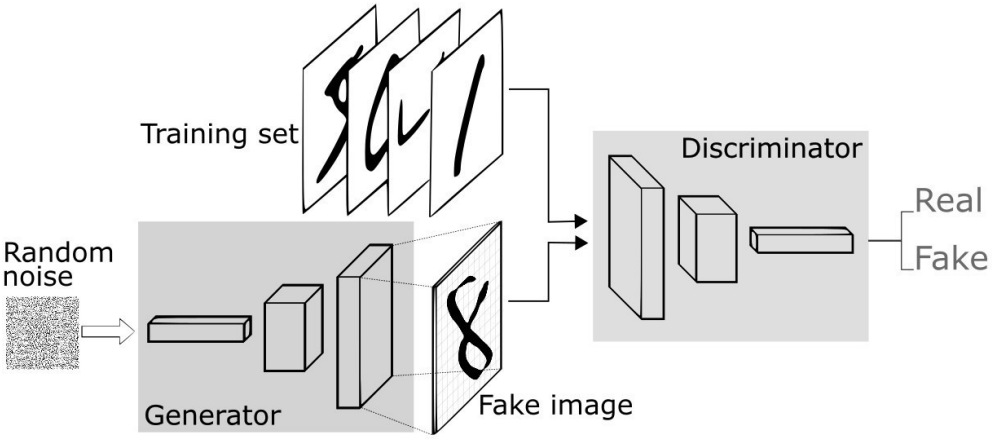


Figure 2: **Generative Adversarial Neural Network's Architecture**

Representation & Learning

Because GANs represent a tremendous opportunity for us, knowing what input to show them is crucial. We have the opportunity to let the model learn directly from floor plan images. **By formatting images, we can control the type of information that the model will learn**. For example, simply showing our model the shape of parcels and associated building footprints will yield a model able to create typical building footprints given a parcel's shape. To ensure the quality of the outputs, we will use our own architectural "*sense*" to curate the content of our training sets: a model will only be as good as the data we give it.

To illustrate, below is a typical training sequence (*Figure 3*): this sequence, realized over the course of a day and a half of training, displays how one of our GAN-models progressively learns how to layout rooms and fenestration for a housing unit.

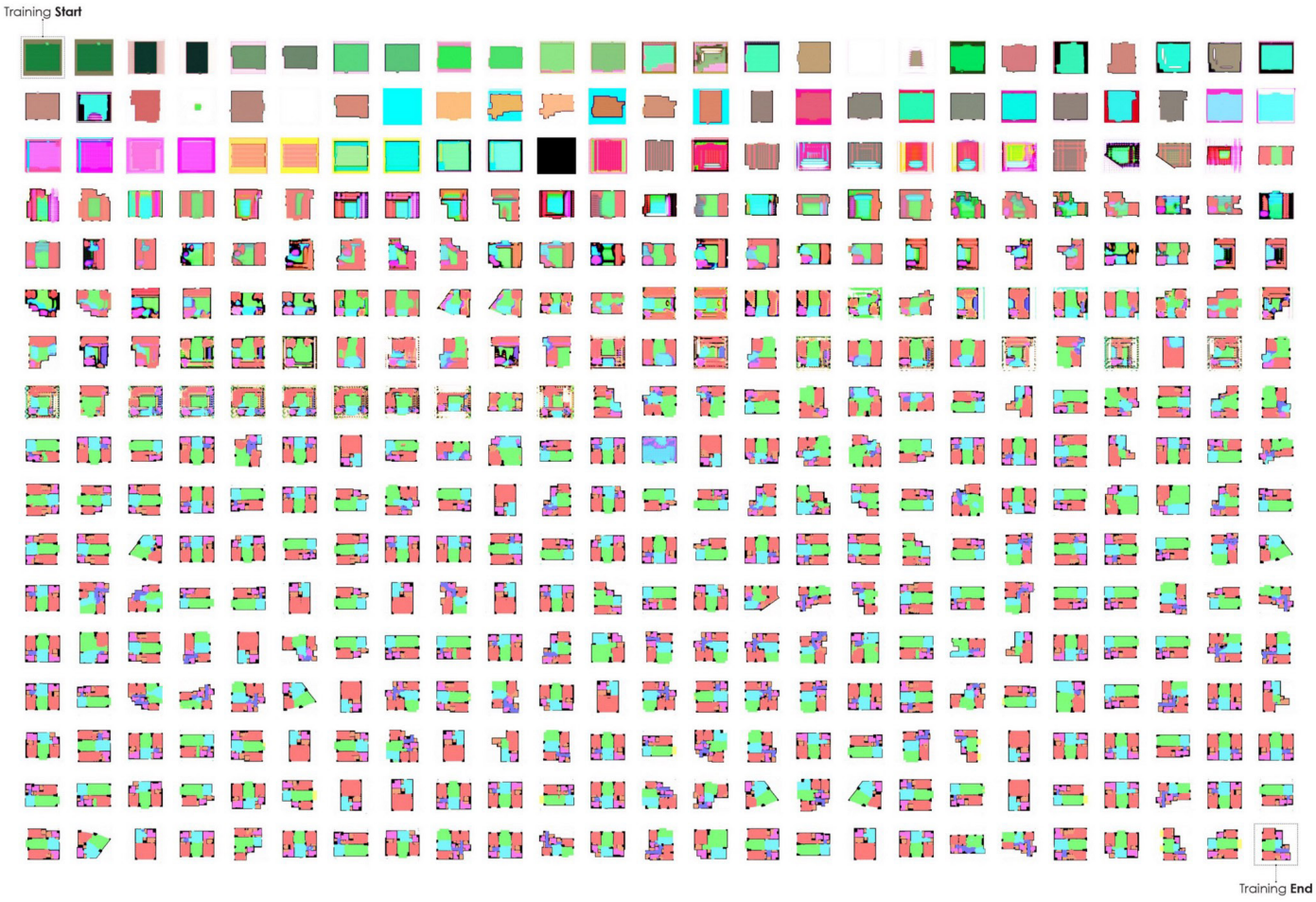


Figure 3: **Training Sequence** | Source: Author

II. Data Curation & Training

We propose to apply GANs to floor plan generation. As the above example shows, GANs, and more specifically Pix2Pix, can be trained to draw room layouts. By feeding pairs of images to Pix2Pix, the model will learn the mapping from one image to the other.

Below, we display images taken out of one of our training sets (Figure 4).

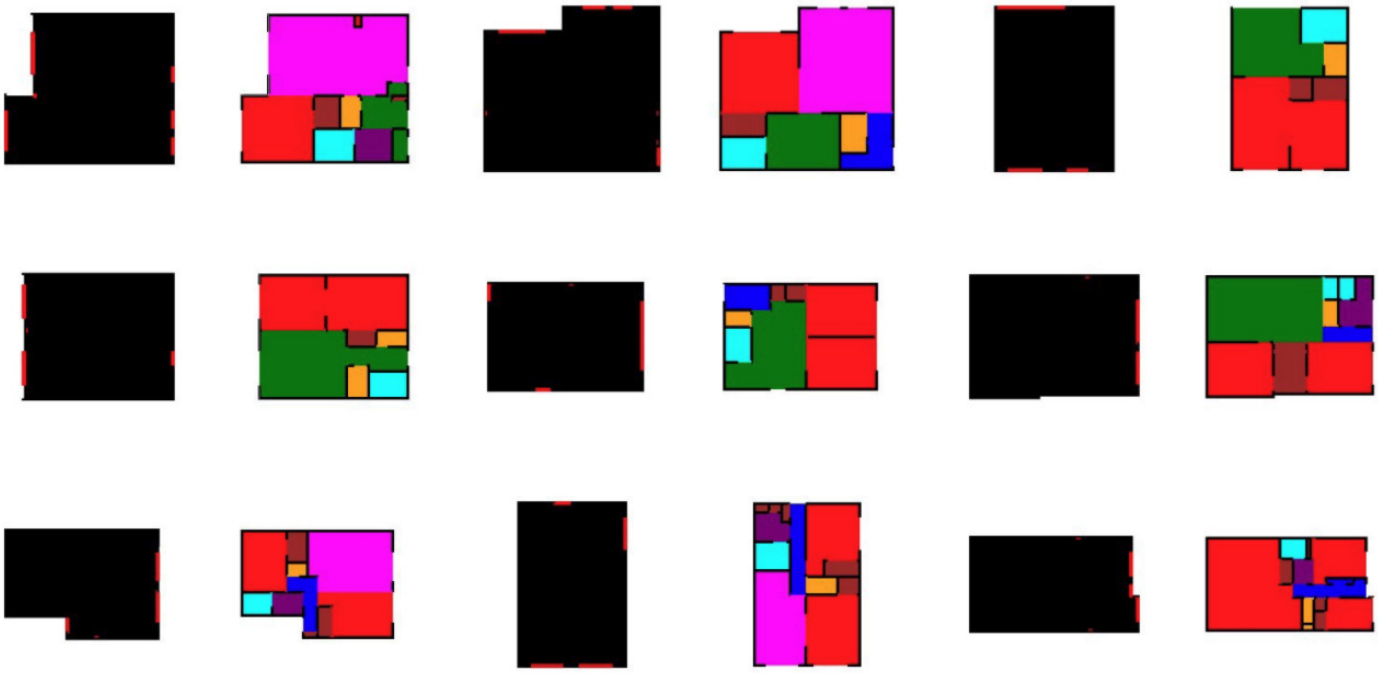


Figure 4: **Sample of Training Set** | Source: Author

Floor plans' color scheme | Source: Author

In the above images, the left-side image (input) encodes the **footprint of the apartment unit in black** and the **position of the facades opening in red**. On the right, the pair displays the corresponding room layout, with room program encoded using a color scheme described above.

. . .

III. Generation

In this chapter, **we showcase three different generation paradigms, all geared towards different realities of the drafting process**. As an architect draws a floor plan, constraints frame his/her design process: the existence of a structural grid, for instance, conditions the placement of walls in space; the necessity of having a given room at a given place puts the entire space layout under stress; the presence of facade opening at a certain spot defines how rooms should be set, etc.

We have isolated 3 main use-cases (*Figure 5*) when it comes to designing housing floor plans, corresponding to varying type constraints.

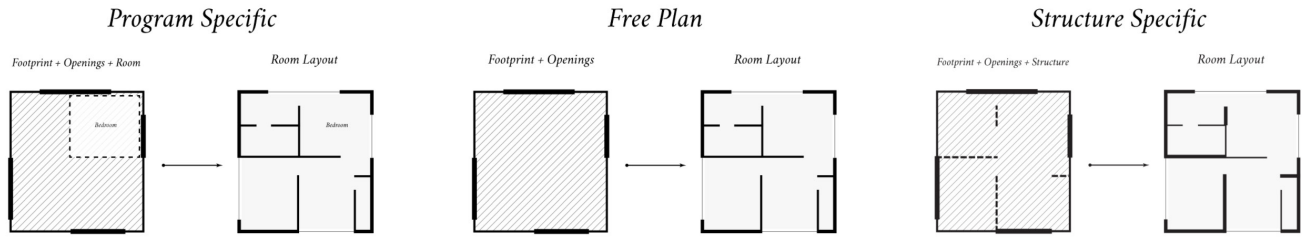


Figure 5: **3 use-cases** | Source: Author

A) *Free plan* *generation* addresses simply cases where the user would only specify :

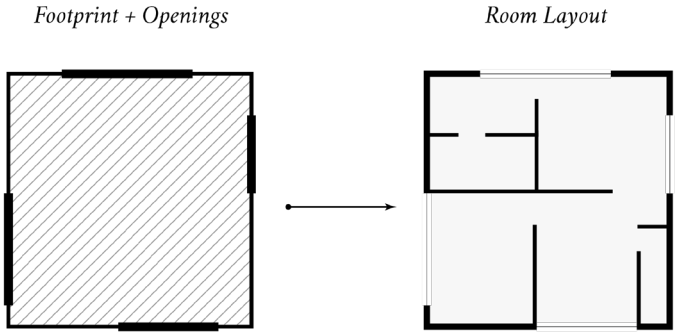
- [1] the footprint of the apartment unit
- [2] the position of the facade's openings

B) *Program-specific* *generation* addresses cases where the design would be driven by the placement of a specific room type. In particular, the user would specify:

- [1] the footprint of the apartment unit
- [2] the position of the facade's openings
- **[3] the position of a given room within the apartment footprint**

C) *Structure-specific* *generation* addresses cases where the design would be driven by

A. Free Plan Generation



In this first approach, **we hope to let our model make most decisions while constraining it with a minimum amount of *internal* constraints.** No structure or room placement will condition the layout of elements in space; the user will only specify the opening in the facade and the apartment footprint. Our GAN model will then *freely* plan out space and layout rooms, walls, and openings between rooms by mimicking properties found among the training set.

Below, we display some typical results (Figure 6).

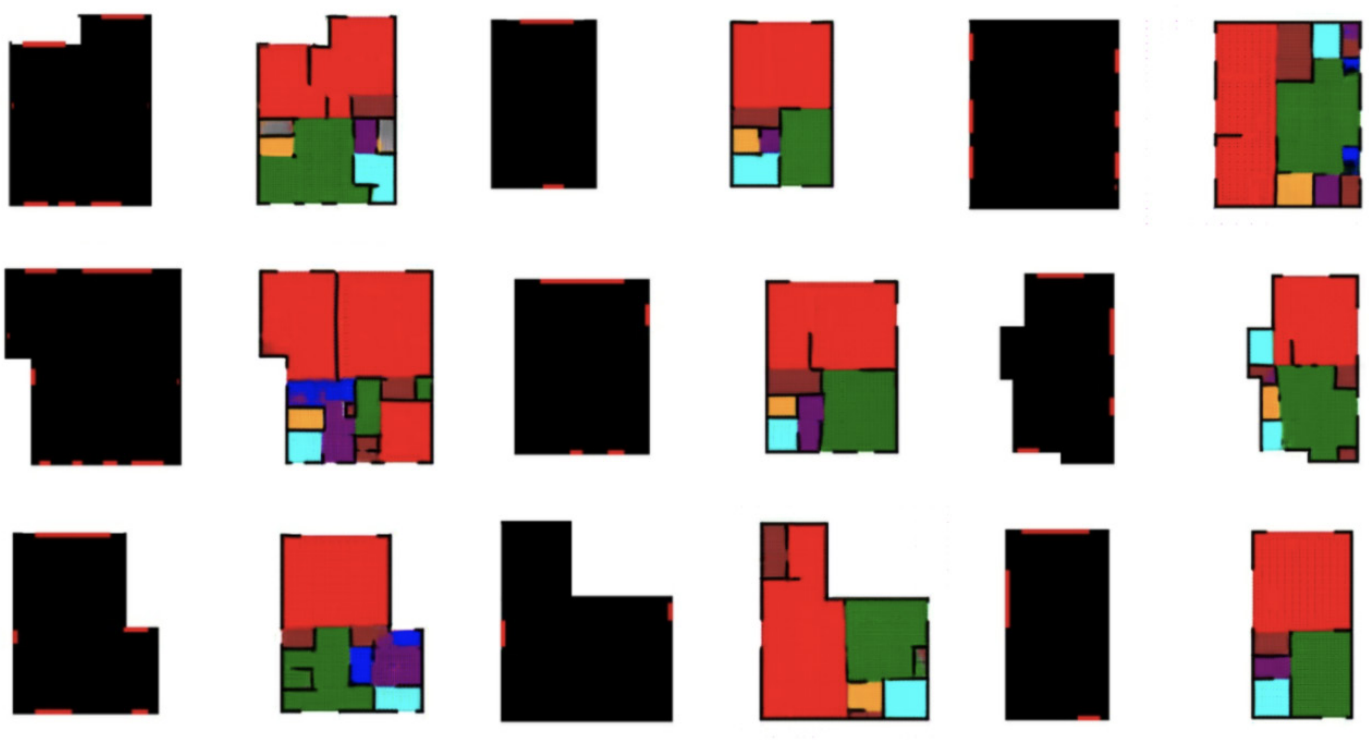
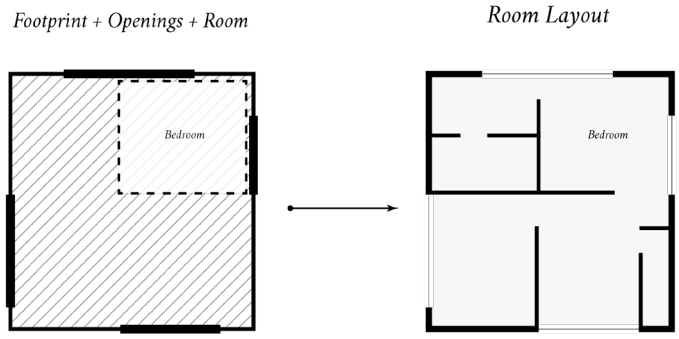


Figure 6: **GAN-generated results** | Source: author

Floor plans' color scheme | Source: Author

. . .

B. Program-Specific Plan Generation



In this approach, **we assume the generation is primarily driven by the placement of a specific room.** For instance, for a given apartment layout, a designer might want to

define where bedrooms would be and leave to the machine the responsibility of placing by itself the remaining rooms. We support this design logic by allowing users to input bedroom size and position in the input (green patch), as well as the apartment footprint and the facade's openings. As a result, the model is able to respect this initial placement, while surrounding bedrooms with "*serving*" spaces (other rooms that should be present in the program).

Below, we display some typical results (Figure 7).

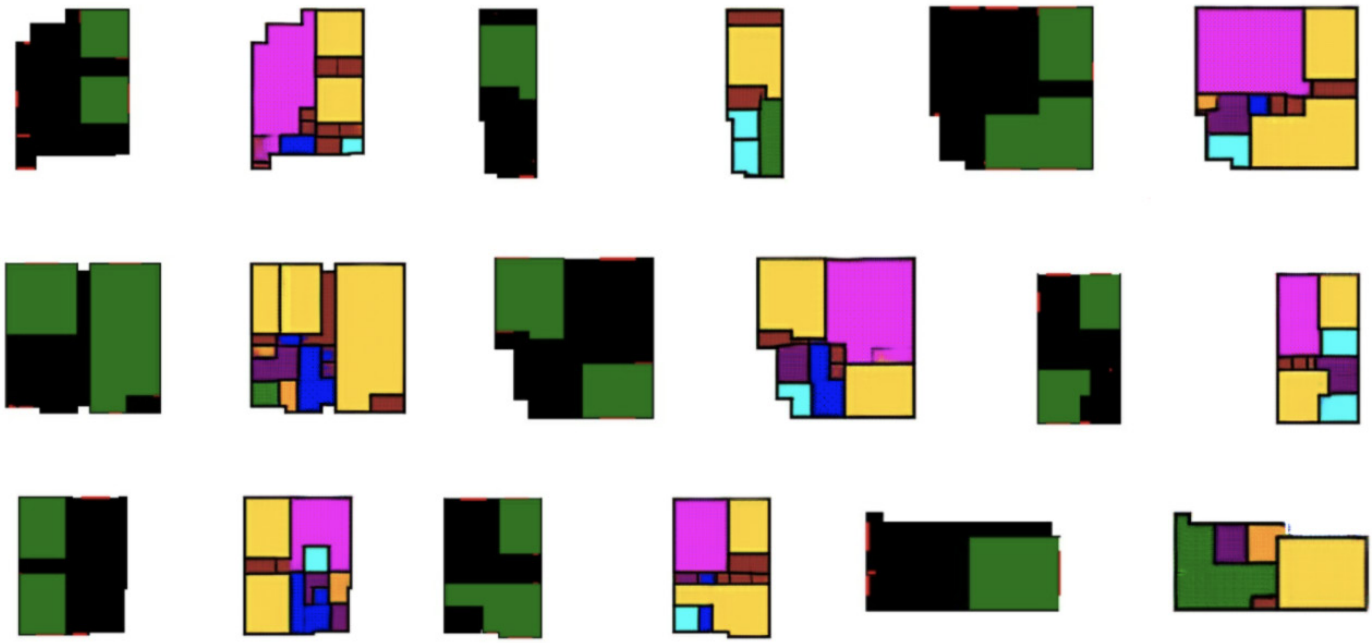
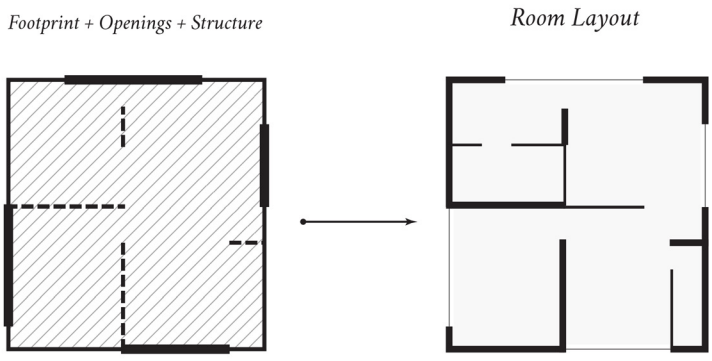


Figure 7: **GAN-generated results** | Source: author

Floor plans' color scheme | Source: Author

. . .

C. Structure-Specific Plan Generation



In the third approach, **we take into account the existence of load-bearing walls as initial constraints.** By marking the input images of the training set with green lines, we signal to our GAN model the presence of such walls and train it to generate room layouts with respect to these structural elements.

Below, we display some typical results (Figure 8).

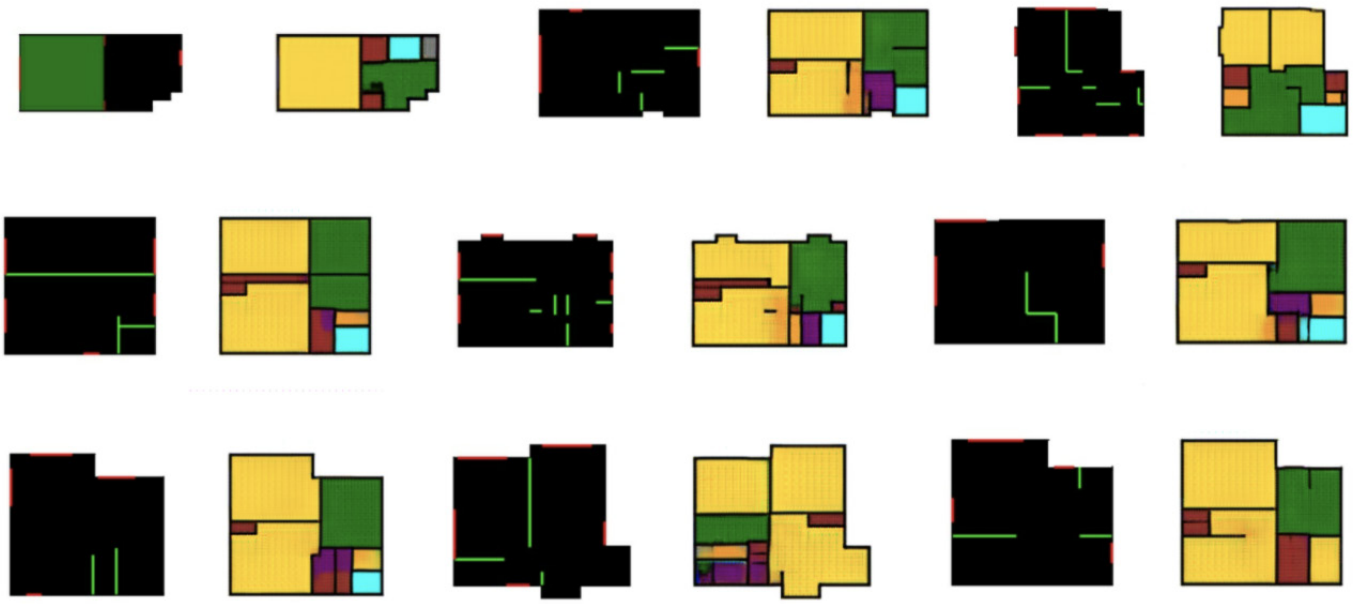


Figure 8: **GAN-generated results** | Source: author

Floor plans' color scheme | Source: Author

. . .

IV. Vectorization

In this final step, **we offer a simple pipeline to vectorize our GAN outputs** (*Figure 9*). To be clear, because raster images are the result of our generation models, we need to post-process these outputs, to turn them into actual geometry. The above illustration describes the overall pipeline.

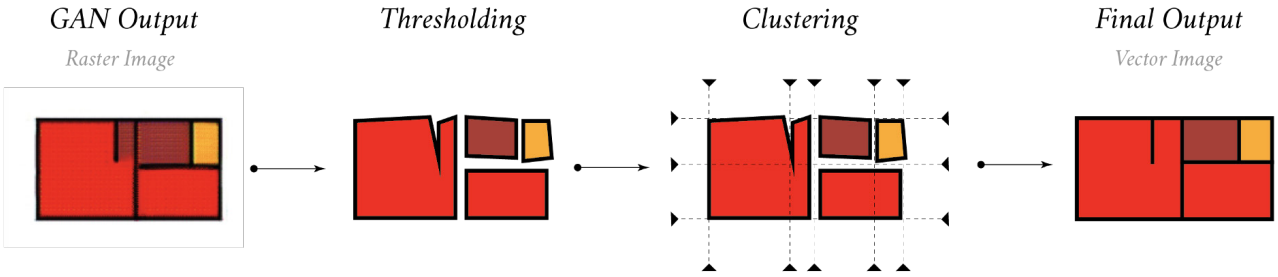


Figure 9: **Vectorization Steps & Pipeline** | Source: Author

Thresholding

We first threshold the initial GAN output to extract the boundary of each room, while keeping track of their respective program. This thresholding can be easily achieved using room colors and contour selection in OpenCV. The resulting outline, however, is wiggly and inaccurate, but the program type and the position in space of each are respected.

Clustering

Next, **we restore the orthogonality of each room polyline**, while snapping outlines together using an X-Mean clustering algorithm. *X-Mean clustering* simply checks the distribution of points along the X and Y directions and creates clusters of these points in an unsupervised fashion. Each cluster will then yield a "grid line" to which any neighboring point will be snapped.

Results

Below, we display the results of our vectorization pipeline (*Figure 10*). If the floor plan shape and the relative placement of rooms are kept in the process, some finer details can be lost along the way.

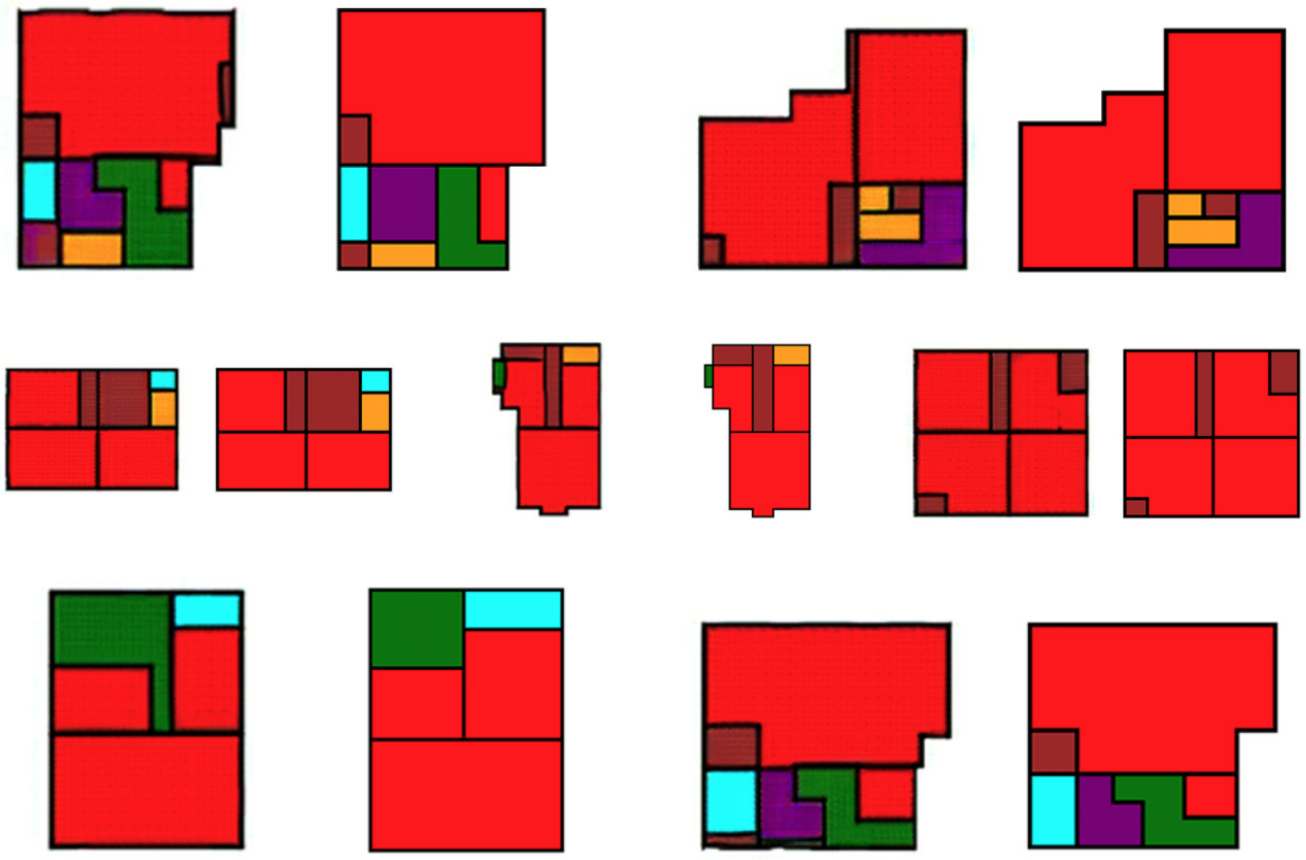


Figure 10: **Results of Vectorization Steps** | Source: Author

Floor plans' color scheme | Source: Author

V. Conclusion

The generation of apartment layouts can tremendously benefit from a statistical approach. When drawing floor plans, GANs seem better suited to handle complex topological problems, and can potentially outperform, or at least complement, previous procedural techniques.

In this article, **we have showcased only some variations around this topic, and hope to see more investigations in this area in the near future.** The latest developments in graph generation using GANs open alternative techniques to generate similar results, while retaining more control over the geometry being generated than with raster-based GAN techniques.