Contour detection

Fabien Lagnieu, Maha Kadraoui, Tristan Waddington, Abdoul Zeba
Mentor: Stéphane Maviel (CEO VINCI/DIANE[1])

Architects are drawing floor plans using a Computer-Aided Design (CAD) software. These files are used by engineers to plan and conduct the construction of the building. In this process, additional measures are needed such as the room size or adjacency. **The goal of this project is to develop a tool that can automatically detect the contours of the rooms in the floor plan to draw it back into the CAD software.**

1 Context

VINCI's engineers are dealing with a different floor plan for every project they are working on. Each plan has its own drawing style and the contours of the rooms are often mixed with other lines. But the contours are needed to plan the location of smoke detectors, fire extinguishers, and other facilities in the building. The goal of this project is to develop a tool that can automatically detect the contours of the rooms in the floor plan and draw back the polygons in the CAD software as a vector layer for further automatic processing. DIANE's teams have already worked on this problem and have developed a first version of the tool that is too slow and not accurate enough. They are looking for other approaches to improve it.

Datasets We are given examples of GeoJSON[2] exports of CAD plans. The first dataset is composed of 9 GeoJSON exports of floor plans from very different buildings (datacenter, hospital, car-park ...). Each file contains a list of points connected together to form lines or polygons representing walls, windows, doors, or even furniture.

The second dataset exports the floor plans of 5 whole buildings in the same format as the previous one. Each plan is joined by a second file containing the actual contours of the rooms as a target. This dataset is really useful to compute metrics of our models.

2 Methods

After a first phase of data cleaning we focus on two different paths to solve the problem:

1. **Geometric process** Transform the geometries into normalized segments [4] then cluster the lines to create the polygons[1] of the contours. This path relies on the geometric libraries `shapely` and `geopandas`. This knowledge was precious to return the detected rooms in the correct format. The graph representation was later discarded as too computational expensive and not precise enough in our clustering methods.

2. **Image segmentation** Using a dense literature review[3], we spot some successful uses of CNNs[5]. This same paper also provides the list of datasets of floor plans as images we

[1]DIANE is VINCI-ENERGIES's inside Startup tasked to find numerical solutions to ease the work of VINCI collaborators.

[2]The current standard for GeoJSON format was published in August 2016 by the Internet Engineering Task Force (IETF), in RFC 7946.

can use for training deep learning vision-based models. Ultimately, this path gives us the idea of to transform the floor plans into images to be segmented into rooms, using basic computer vision methods and the Segment Anything Model (SAM)[2].

At the end of our research, we produce 3 models of different complexity, staking together like matriochkas, ranging from geometric hand-crafted rules to fully automatic computer vision: `SegmentBasedClustering`, `VisionSegmentation`, and `SegmentAnythingModel`. We started with a simple model and successively add complexity at some points in the pipeline expecting to improve the results. Models share common steps and a global representation is given in figure 1. The individual pipelines are detailed in the following sections.

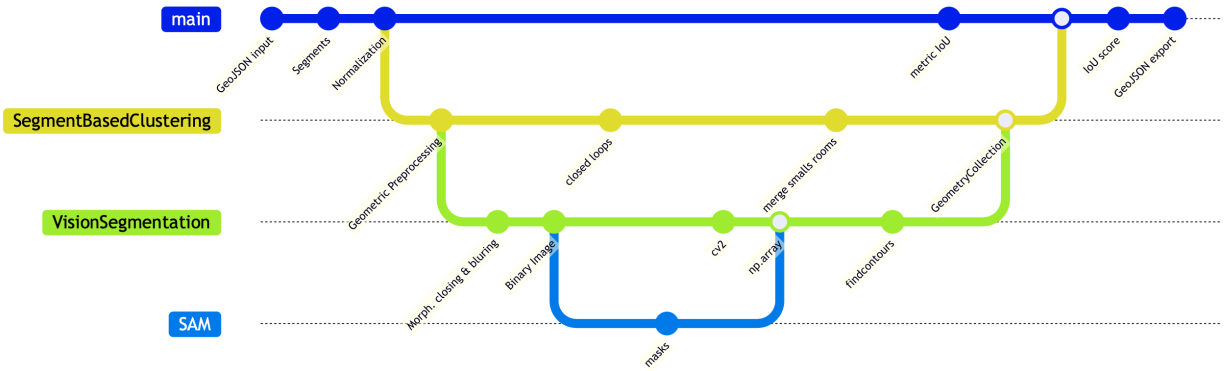


Figure 1: Articulation of the shared steps and imbrication of the three models.

2.1 Geometric Preprocess

Before feeding GeoJSON files into the model, a preprocessing step is applied to clean and standardize the segment data. This ensures accurate room segmentation while minimizing errors caused by noise, misaligned geometries and above all, plans drawn with different scales.

Preprocessing Steps The preprocessing consists of five successive steps: (i) **Splitting into segments** the complex geometries. (ii) **Normalizing segments** by sorting segment endpoints to avoid duplicate representations and rescaling to the meter unit. (iii) **Snapping** close endpoints by aligning nearby segment endpoints (within 0.05m) for better connectivity. (iv) **Removing isolated** segments by eliminating segments not connected to any others. (v) **Merging collinear** segments by combining nearly aligned segments to reduce fragmentation.

The preprocessing significantly reduces the number of segments (up to 70%) while preserving the overall geometry. See figure 5 for visual results. Additional details and tables can be found on appendix A.

2.2 Model 1: `SegmentBasedClustering`, Based on Geometric Rules

This model is a geometric rule-based method designed to extract room boundaries from architectural floor plans provided in GeoJSON format. This method leverages classic computational geometry techniques to detect closed paths and segment parts based on structural constraints. It does not require training data. Further details and pipeline are described in appendix B.

Discussion `SegmentBasedClustering` offers an efficient approach to segmenting rooms in architectural plans without resorting to complex models nor heavy computing resources. By leveraging topological and geometric operations, the method ensures robust room segmentation. How-

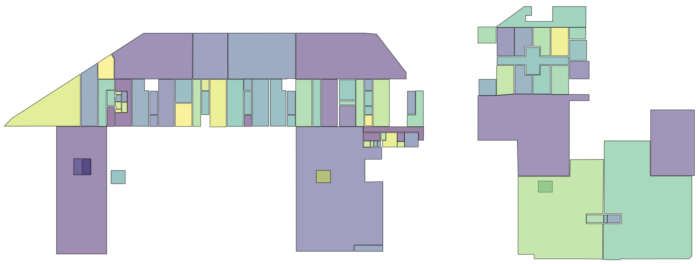


Figure 2: Example of precise rooms detections using `SegmentBasedClustering`.

ever, this approach needs explicit physical separations between rooms. If there are no walls or voids, inconsistent results may be obtained.

2.3 Model 2: `VisionSegmentation`, Based on Computer Vision

The other rule-based method leverages classical computer vision techniques to extract room boundaries from geometric data, without requiring training datasets.

The pipeline consists of converting vector data into raster images, applying image processing operations to detect closed areas, and exporting the resulting room contours back into a geo-referenced vector format (GeoJSON). The entire workflow is based on `OpenCV` image processing primitives and geospatial data transformations from `shapely`. Further details and pipeline description can be found in appendix C.

Figure 3: Example of rooms detections using `VisionSegmentation`. Left: Binary image generated from vector segments. Right: Colorized visualization of detected rooms. Room without doors are merged in this version.

Discussion This vision model offers a simple yet robust solution for room segmentation on vector floor plans. It avoids the complexity of enforcing strict topological rules in vector space and instead leverages the flexibility of raster-based computer vision techniques. The method is particularly suited to noisy, exports where traditional vector-based approaches may struggle.

However, the `VisionSegmentation` also depends on the explicit representation of physical separations. In cases where doors are missing, or open connections are not physically modeled, adjacent spaces may be incorrectly merged into a single room.

2.4 Model 3: `SAM`, Based on Segment Anything

To handle the open door issue, we explore another vision-based model based on the Segment Anything Model (SAM)[2]. SAM is a foundation model for image and videos segmentation. Developed by Meta's FAIR (Fundamental AI Research) lab, SAM represents a significant advancement in computer vision, designed to serve as a versatile framework for various segmentation tasks. His architecture can be summarized in three parts: a Vision transformer image encoder, a prompt encoder, and a modified mask Transformer decoder block (see Figure 8, taken from original paper [2]). Further details and pipeline description can be found in appendix D.

Discussion This model is way more computationally expensive than the two previous ones, and requires GPU memory to detect rooms in seconds. It correctly splits the non-connected rooms and is able to detect rooms without doors. However, the model is not as precise as the other ones.

3 Results

3.1 Construction of a Metric

The classical metric for segmentation tasks is the Intersection over Union (IoU). But in the specific context of room detection, we define a **matched IoU metric** that considers the intersection over union of the detected room with the ground truth room that matches the most with the detection. It computes a score from 0 (no match) to 1 (perfect overlapping). This adaptation is further described in appendix E.

3.2 The Results

Figure 4 displays the evaluation of 2 models on a single plan. The darker the green the better the score of the individual rooms. This metric penalizes our model when it splits a merged room from the ground truth despite a wall drawn in plan. Furthermore, our models do not create rooms open to the sides of the image as a consequence of application of expert rules.

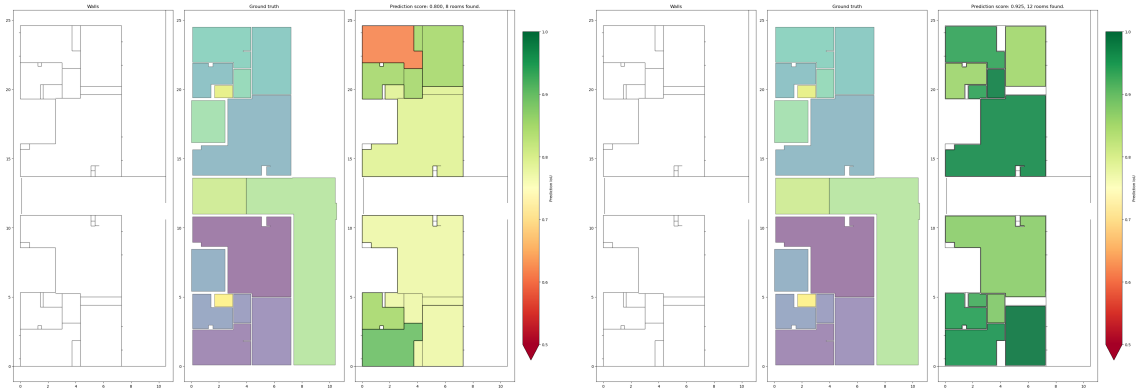


Figure 4: Visualization of original walls / predictions / scoring of the two first models on the same plan (left: `SegmentBasedClustering`, right: `VisionSegmentation`).

Additionally, we conduct a comparison of the three models on the second dataset. Table 1 summarizes the final results and shows a trade-off for `VisionSegmentation`.

Our models have various performances over the different plans. The two first models are very sensitive to the open doors but are more precise and faster than the last one. The last one is more robust but less precise and more computationally expensive.

Model	Mean IoU	Mean Matched IoU	Time
`SegmentBasedClustering`	0.04	0.76	6.8s
`VisionSegmentation`	**0.09**	**0.86**	**4.6s**
`SegmentAnythingModel`	0.06	0.63	2358s

Table 1: Comparison of the three models on the second dataset (best in bold).

4 Conclusion

Main findings This project is amazing in a sense that it is very easy to understand but very hard to solve when digging into it. We really enjoyed the freedom given by VINCI/DIANE to explore different paths and to come up with different solutions. We have also benefited from their expertise to guide us in the right direction and avoid death traps. Our team is pretty proud to have managed to produce three complementary models that are actually able to detect the contours of the rooms in the floor plans. They are not perfect, but they work and respect the requested workflow.

Limitations The models are sensitive to the representation of the plans. Furthermore, we were not able to find an efficient algorithm to close the doors and other corridors. An other improvement would be to discriminate wide walls that are sometimes identified as rooms.

Organization within the team The project initially started in the office of VINCI/DIANE for a brainstorming session. Since then, we have been working remotely in a smooth way, meeting with VINCI/DIANE at the beginning of each Thursday afternoon. We were using a shared git repository to share our code and results[3].

After a first quick diverging phase where every one has tried different methods and dug into the data, we have split in teams of two to focus on two paths described in section 2. Tristan took care of the global information management process and organized the relations with VINCI.

- Tristan handled the initial data cleaning, then worked with Maha who expertly created the geometric process and the segment preprocessing and clustering.
- Abdoul explored first the computer vision path and Fabien made the breakthrough in the results with the image segmentation.

References

[1] Bernardino Domínguez, Ángel-Luis García-Fernández, and Francisco Feito. Semiautomatic detection of floor topology from cad architectural drawings. *Computer-Aided Design*, 44:367–378, 05 2012.

[2] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.

[3] Pablo N. Pizarro, Nancy Hitschfeld, Ivan Sipiran, and Jose M. Saavedra. Automatic floor plan analysis and recognition. *Automation in Construction*, 140:104348, 2022.

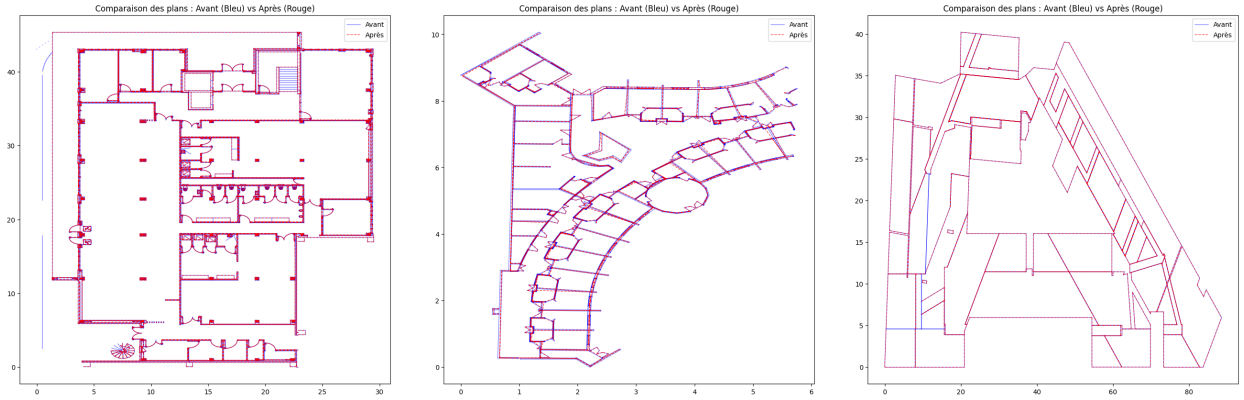
[4] Martin Schafer, Christian Knapp, and Samarjit Chakraborty. Automatic generation of topological indoor maps for real-time map-based localization and tracking. *2011 International Conference on Indoor Positioning and Indoor Navigation*, pages 1–8, 2011.

[5] Jaeyoung Song and Kiyun Yu. Framework for indoor elements classification via inductive learning on floor plan graphs. *ISPRS International Journal of Geo-Information*, 10(2), 2021.

[3] Git Repository: waddason/CapstoneContour

A Additional details on the geometric preprocessing

Preprocessing Results This preprocessing step is particularly effective for large datasets with high redundancy, as it reduces computation complexity while maintaining architectural integrity. But it can be problematic when applied to smaller input datasets like in our third example. So it should be used with caution, as excessive modification of small datasets can worsen distortions, as seen with 520 segments where a 0.443m displacement occurred. As a consequence, the preprocessing step is an option when using our models.



Case1: output2 Case2: output7 Case3: output3

Figure 5: Comparison of floor plans before (Blue) and after (Red) preprocessing for 3 different floor plans. The near absence of blue segments proves that we have removed a lot of redundancy.

The following tables summarize the impact of the geometric preprocessing on the segment count and the average displacement after preprocessing. We used three different floor plans that spans along the range of complexity of the dataset.

The drop in segment count is significant, with a reduction of over 70% in most cases, reducing computational cost of the ulterior tasks. See table 2 for details.

Test Case	Nb segments before	Nb segments after	Reduction %
Case 1	22,425	6,417	71.38%
Case 2	6,013	1,648	72.59%
Case 3	520	330	36.54%

Table 2: Impact of geometric preprocessing on segment count.

This drop in segment count comes with a light cost in the accuracy of the representation. On most cases, the segments are moved of about 2 cm, which is acceptable for the purpose of the project. See table 3 for details.

Test Case	Avg. Displacement
Case 1	0.02
Case 2	0.025
Case 3	0.443

Table 3: Average displacement after preprocessing.

In our code, this preprocessing is done with the method `preprocess_segments .complete_preprocessing(segments, angle_tol=0.1, distance_tol=0.3, snap_tol=0.05) -> cleaned_segments`. Where `angle_tol` is the maximum angle between two segments to be considered collinear, `distance_tol` is the maximum distance between two segments to be merged, and `snap_tol` is the maximum distance between two endpoints to be snapped together.

B Additional details about SegmentBasedClustering model

Pipeline Description The pipeline follows a sequence of geometric operations performed using the `Shapely` computational geometry library.

1. Extracting relevant **wall segments** from GeoJSON data.
2. Identifying **closed loops** using geometric operations.
3. Retaining valid rooms based on predefined **area constraints**, considering only polygons between 1 m^2 and 1000 m^2 (hyperparameters).
4. Identifying which rooms share **common boundaries** by doing adjacency analysis to facilitate merging and spatial organization.
5. **Combining and merges small rooms** (under 5 m^2) with their largest adjacent neighbor.

C Additional Details about `VisionSegmentation` Model

Pipeline Description The method is structured in the following key steps:

1. **Vector Preprocessing.** Raw GeoJSON data often includes heterogeneous geometries (Polygons, MultiPolygons, LineStrings), many of which are noisy or incomplete. We filter the data to retain only the linear primitives representing walls. The geometries are rescaled and recentered to ensure consistent metric resolution before conversion. This corresponds to the steps (i) and (ii) described in the geometric preprocessing in section 2.1.
2. **Binary Image Generation.** The preprocessed segments are rasterized into binary black-and-white images, where each segment is drawn with a configurable thickness (default: 3 pixels). A DPI value of 50 (pixels per meter) ensures sufficient spatial resolution. To improve the connectivity of wall segments, we apply Gaussian dilation, which smooths the edges and *ensures continuity* even in curved or oblique wall segments. Result is shown on figure 3 left.
3. **Morphological Processing and Contour Detection.** A morphological closing operation (`cv2.morphologyEx` with `MORPH_CLOSE` operation) fills small gaps between wall segments. Contours are then extracted with `cv2.findContours` using the `RETR_CCOMP` mode, producing a hierarchical tree structure that allows holes (e.g., internal courtyards or voids) to be identified and processed correctly.
4. **Wall Filtering and Room Selection.** To discriminate between rooms and walls, we apply geometric criteria. Only contours with a surface area greater than 1 m^2 (2500 pixels at 50 DPI) and an average thickness (surface-to-perimeter ratio) exceeding 0.4 meters are retained. Best parameters displayed in table 4 have been found using the validation dataset.

   Additionally, the longest contour—typically representing the building envelope—is discarded. The detected rooms are shown on figure 3 right.
5. **Polygon Conversion and GeoJSON Export.** The valid contours are converted back into metric coordinates using transformation parameters from the rasterization step. Polygons are enriched with properties such as area (m^2) and a unique room identifier. Final outputs are exported in GeoJSON format to be loaded back into CAD software.

Hyperparameters tuning An extensive hyperparameter search was conducted to optimize the model's performance. The best parameters are summarized in table 4.

Numerous tests were conducted using the validation test as a benchmark and using the matched IoU score to select the best parameters. Example of experiments during the early stage of the project are shown in figure 6. The DPI value of 50 pixels was kept, but Fabien managed to optimize the code to reduce the thickness from 7 to 3 pixels only. This produced contours closer to the actual walls.

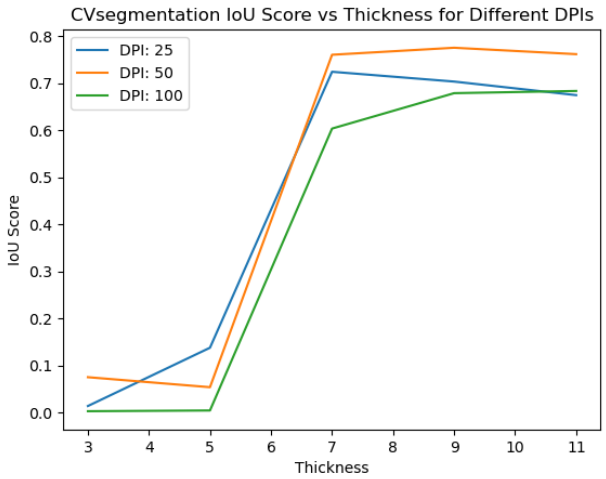


Figure 6: Example of hyperparameters tuning during the early stage of the project.

Ultimately, Fabien developed a dynamic interface to visually check the effects of the parameters on the segmentation. This tool was used to fine-tune the parameters and to define their default values. An illustration of the interface is depicted bellow in figure 7.

DPI	50 pixels per meter (1 pixel ≈ 2 cm)
Wall thickness	3 pixels (default)
Minimum room area	1 m^2
Minimum average thickness	0.4 m
Dilation method	Gaussian

Table 4: Best `VisionSegmentation` Model Parameters

D Additional Details about `SAM` Model

Pipeline Description We first produce a high-quality image using the two first steps of the `VisionSegmentation` pipeline presented earlier. Then, we use SAM to infer masks on the image as segmentation of all the rooms. This step requires access to GPU. Finally, the contours of the masks are extracted, and after rescaling the contour coordinates, we generate the corresponding GeoJSON file. The SAM checkpoint used is described in table 5.

Due to the high computational cost of the model, we have not been able to conduct an extensive hyperparameter search. Partial results about the best thickness are shown in figure 9.

E Additional Details about the Intersection over Union Metric

During their extensive review of the literature, the Chillian team of Pizarro et al.[3], found several possible metrics to evaluate the performance of the models. Typically, segmentation results were

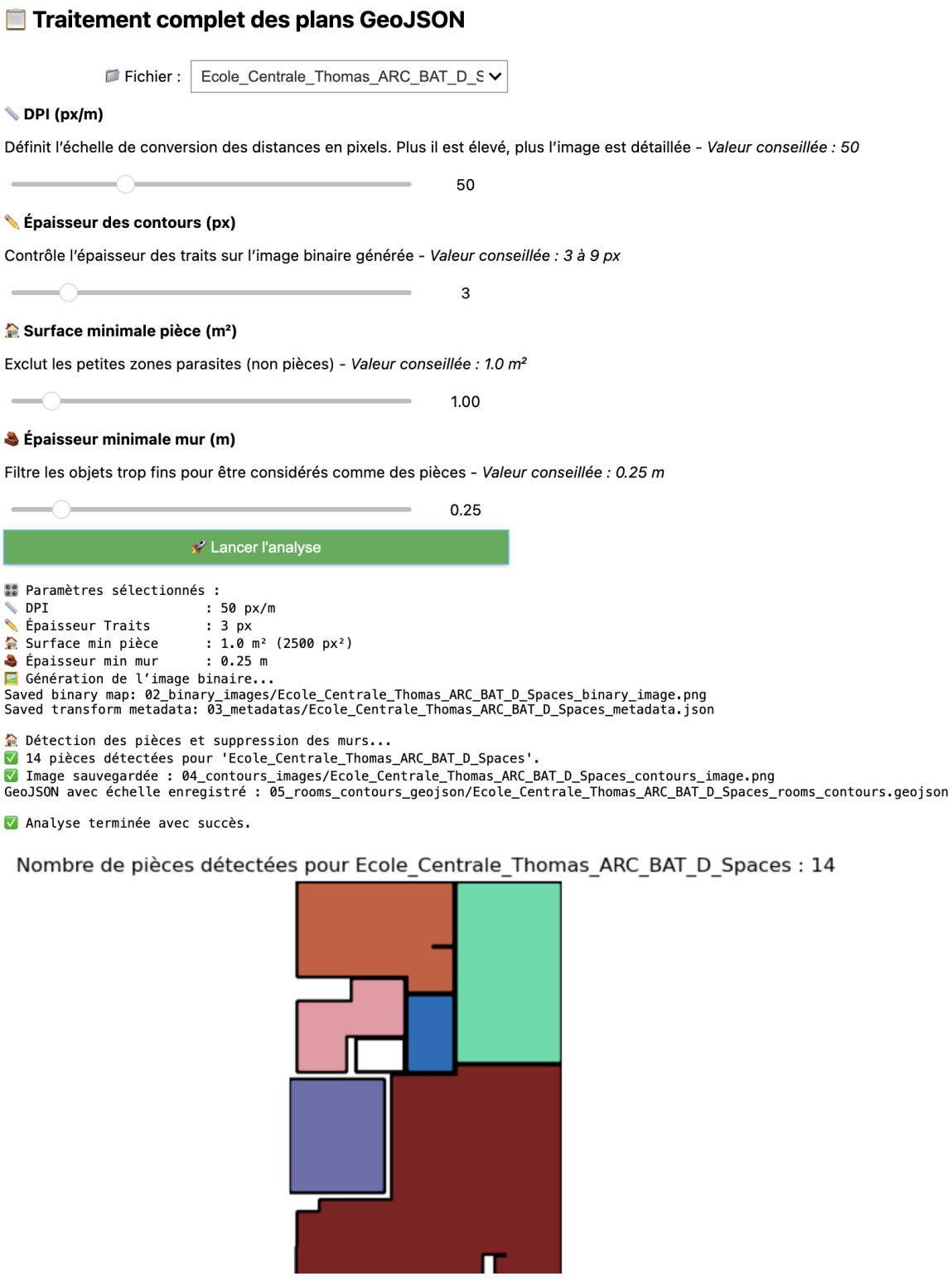


Figure 7: Dynamic interface to fine-tune the parameters of the `VisionSegmentation` model, launched inside a Jupyter notebook.

Checkpoint	`sam_vit_h_4b8939.pth`
Backbone Size	Huge (ViT-H)
Model Parameters	636 million
File Size	2.4 GB
GPU Memory Requirement	At least 8 GB
Use Case	Tasks requiring high-quality segmentation

Table 5: `SAM` Model Checkpoint Information

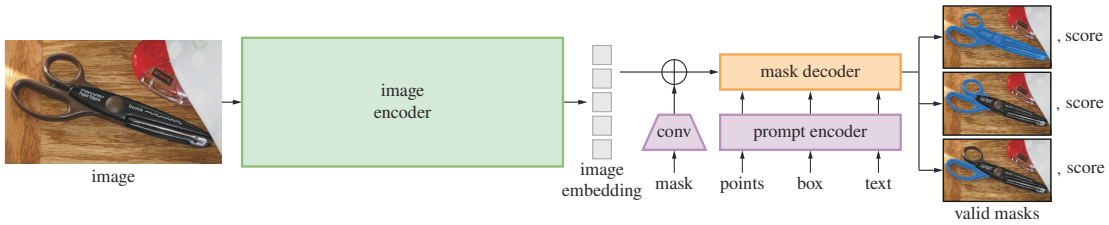


Figure 8: Segment Anything Model (SAM) overview

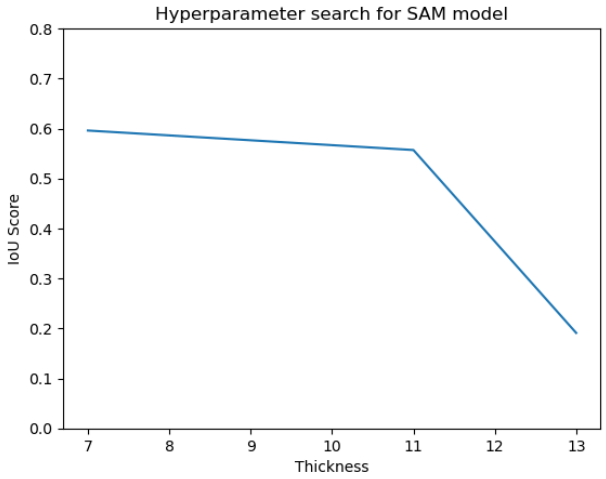


Figure 9: Example of hyperparameter tuning for `SAM` model focusing on the thickness of walls.

evaluated in terms of the intersection over union (IoU), pixel/class accuracy, and the Jaccard Index (JI) proposed by de Las Heras et al.. By contrast, works that detected objects (e.g., walls, doors, windows) used the mean average precision (mAP), the recall & precision, the match score (MS), detection rate (DR), and recognition accuracy (RA), or considered a confusion matrix. Hence, we decide to focus on the IoU metric.

E.1 IoU for Object Detection

Definition The Intersection over Union (IoU) metric is a standard measure of the accuracy of a segmentation model. It is defined as the ratio of the area of intersection to the area of union of the Ground Truth (GT) and the segmentation mask (S). The mathematical formula is given below, and an illustration is provided in figure 10.

- True Positive: The area of intersection between Ground Truth (GT) and segmentation mask (S). Mathematically, this is logical AND operation of GT and S i.e. $TP = GT.S$,
- False Positive: The predicted area outside the Ground Truth. This is the logical OR of GT and segmentation minus GT. $FP = (GT + S) - GT$,

- False Negative: Number of pixels in the Ground Truth area that the model failed to predict. This is the logical OR of GT and segmentation minus S $FN = (GT + S) - S$,
- Final IoU: The ratio of the area of intersection to the area of union of GT and S. Mathematically, $IoU = \frac{TP}{TP+FP+FN}$

Figure 10: IoU is the ratio of the intersected area to the combined area of prediction and ground truth masks. (source: learnopencv.com)

In our code `utils.metric.intersection_over_union(gt, pred) -> float`, compute the score using the ratio of the calculated areas thanks to the powerful `shapely` library that provides `intersection` and `union` methods for polygons.

E.2 Matched IoU for Room Detection

Definition In the context of room detection, we define a **matched IoU metric** that considers the intersection over union of the detected room with the ground truth room that matches the most.

Initially, the metric is computed as the average IoU over all detected rooms. Then an optimization algorithm is used to match the detected rooms with the ground truth rooms to maximize the average IoU. Specifically, we used the Hungarian algorithm from Harold Kunhn to solve the assignment problem, using the `scipy.optimize.linear_sum_assignment` function. The final metric is the average IoU of the matched rooms. Despite its complexity, this metric is more informative than the simple IoU metric and is more adapted to score our models as the results are closer to 1 (the optimal score) than when averaging over all rooms.

Possible extentions We choose not to penalize the non-detected rooms, because we did not have to train models from scratch. Ultimately, we also define a Jaccard loss in the purpose of training future models. But it is not used in the current project.

F Additional Segmentation Results

The figure 11 next page displays the segmentation results of the three models on random floor plans. The first line shows the results of the `SegmentBasedClustering` model, the second line shows the results of the `VisionSegmentation` model, and the third line shows the results of the `SegmentAnythingModel` model, each with a distinctive color map.

The left column shows the prediction on the same plan for comparative purpose. On this simple well drawn example, all models correctly detects the rooms and corridors. We note that the `VisionSegmentation` model get the best results on complex plans (center).

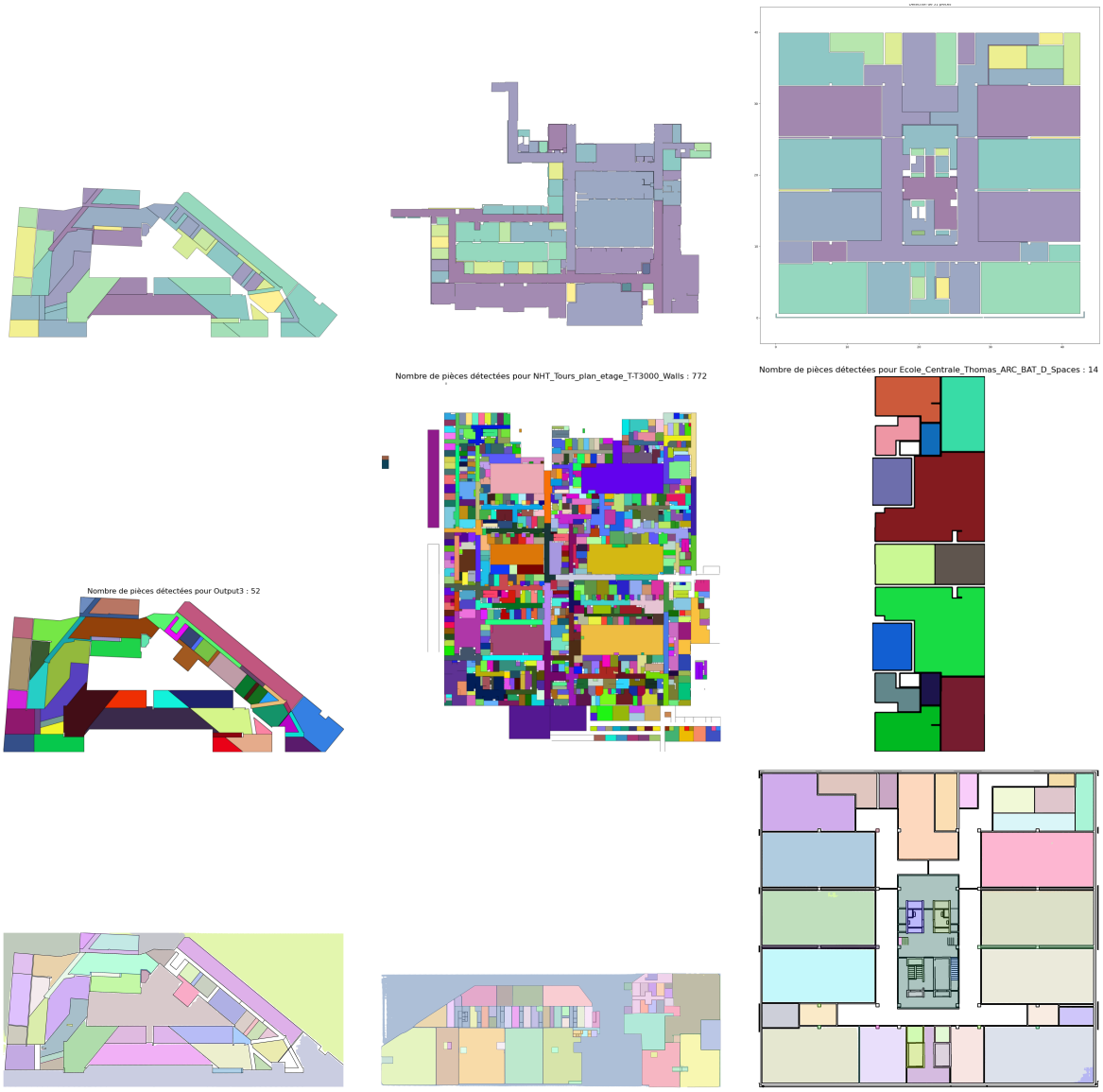


Figure 11: Segmentation outputs for random floor plans. First line: results of the `SegmentBasedClustering` model. Second line: results of the `VisionSegmentation` model. Third line: results of the `SegmentAnythingModel` model.