WAFFLE: Multimodal Floorplan Understanding in the Wild –Supplementary Material–

A. Interactive Visualization Tool

Please see the project page (https://tau-vailab.github.io/WAFFLE) for an interactive visualization of data from the *WAFFLE* dataset.

B. Additional Dataset Details

We proceed to describe the creation of our *WAFFLE* dataset in the sections below, including details on curating, filtering, and generating pseudo-ground truth labels.

B.1. Model Checkpoints and Settings

We use Llama-2 [9] for text-related tasks, and CLIP [7] for image-related tasks. For most text related tasks we use the `meta-llama/Llama-2-13b-chat-hf` model, and for legend extraction we use the `meta-llama/Llama-2-70b-chat-hf` model. In both cases, we use the default sampling settings defined by the Hugging Face API. For CLIP, we use the `openai/clip-vit-base-patch32` model.

B.2. Layout Component Detection

As part of the data collection, we train a DETR [1] object detection model to identify common floorplan layout components which will be later on used for the pGT extraction and in the segmentation experiment. We use the checkpoint `TahaDouaji/detr-doc-table-detection`[*] as the base model, and fine-tune it on 200 manually annotated images with augmentations, using the following labels: `floorplan, legend, scale, compass`. We fine-tune for 1,300 iterations, a batch size of 4, and a 10^{-4} learning rate on one A5000 GPU, splitting our data into 80% training images and 20% test images.

B.3. Data Filtering

As described in the paper, we first scrape a set of images and metadata from Wikimedia Commons and proceed to filter to only select images of floorplans using a two-stage process: text-based filtering with an LLM, and image-based filtering with CLIP. All models used for these stages are described in our main paper.

[*]Denotes equal contribution

[*]https://huggingface.co/TahaDouaji/detr-doc-table-detection

Prefixes	"an illustration of ", "a drawing of ", "a sketch of ", "a picture of ", "a photo of ", "a document of ", "an image of ", "a visual representation of ", "a graphic of ", "a rendering of ", "a diagram of ", ""
Negative Prefixes	"a 3d simulation of ", "a 3d model of ", "a 3d rendering of "
Positive Suffixes	"a floor plan", "an architectural layout", "a blueprint of a building", "a layout design"
Negative Suffixes	"a map", "a building", "people", "an aerial view", "a cityscape", "a landscape", "a topographic representation", "a satellite image", "geographical features", "a mechanical design", "an engineering sketch", "an abstract pattern", "wallpaper", "a Window plan", "a staircase plan"

Figure 1. The prompts used for gathering CLIP scores. (Prefixes × Positive Suffixes) are used as positive prompts, and ((Prefixes + Negative Prefixes) × Negative Suffixes) are negative prompts.

B.3.1 Text-based filtering (LLM)

First, we query an LLM in order to obtain an initial categorization of our raw data. We ask it to choose what the image is most likely a depiction of out of a closed set of categories (i.e. multiple choice question format), marked positive (e.g. *floorplan* or *building*) or negative (e.g. *map* or *park*), and we filter out images categorized as negative categories. The full prompt is shown on the leftmost column of Figure 18, where options A and B are treated as positive and the rest are negative.

B.3.2 Image-based filtering (CLIP)

We proceed to use image-based filtering to yield our final dataset. This is composed of two sub-stages: first, we generate a smaller set of highly accurate images (a *seed*); we then extend this seed to produce an enlarged dataset. These sub-stages are described below.

Seed generation. We start by creating a highly accurate seed of images (*i.e.* containing floorplans) by aggressively filtering according to the CLIP normalized scores extracted over positive and negative prompts. We list the prompts used in Figure 1. As illustrated in the figure, negative prompts correspond to images that depict categories similar to floorplans, such as maps or satellite images. We sort the prompts by score, and add images to the set if it passes the following two tests: (i) All top five prompts contain *floor plan*, and (ii) The sum of all prompts containing *floor plan* is over 0.5. We empirically find that these tests allow for creating a highly accurate seed of 3,402 images.

Dataset extension. Next, we use this seed to bootstrap an image classifier, in order to enlarge the dataset. We first use this seed to train a vision transformer binary classifier. We take 1K images from the seed as positive samples, and 1K images that were categorized as a negative category in the text-based filtering step as negative samples. We fine-tune a ViT model [2] for 5 epochs with a batch size of 4 and a $2 * 10^{-4}$ learning rate on one A5000 GPU, splitting our data into 1,400 training images, 300 validation and 300 test images.

To create our final dataset, we select images that pass the following two tests: (i) classifier threshold selected to filter out 10% of data, and (ii) the sum of normalized scores on all positive prompts (described in Figure 1) is over 0.5.

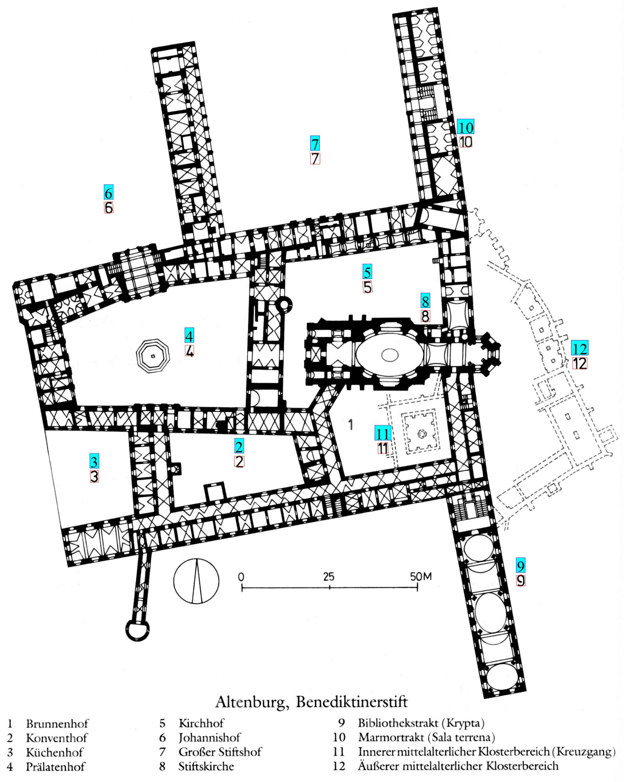
Altogether this leads us to the final dataset of ∼19K floorplans. In addition to the manual validation over 100 random sampled images in the dataset, we also manualy inpect the entire test set, and remove all image that do not contain a valid floorplan. Based on this validation, we estimate that 89% of images from our full dataset are indeed floorplans.

B.4. LLM-Driven Structured pGT Generation

Figure 18 contains the prompts used for extracting the pseudo-ground truth (pGT) labels for our dataset. Note that some prompts use previously extracted pGTs as inputs, such as those for "Building Type" and "Location Information".

The architectural feature grounding process is split into two: legend extraction from the image metadata, and architectural information extraction from the image.

Legend Structuring from Metadata. We divide the task of legend structuring into four sub-tasks: (i) Legend raw content extraction (the raw text containing key–value pairs), extracted using the prompts in Figure 19; (ii) Key–value identification (raw text structurization) using regular expressions on the raw text legend; (iii) Legend content simplification, using the prompt in Figure 20; and (iv) grounding the architectural features in the legend to the image, by



6
7
10
5
8
4
12
1
11
2
3
9
0
25
50M
Altenburg, Benediktinerstift
1 Brunnenhof
2 Konventhof
3 Küchenhof
4 Prälatenhof
5 Kirchhof
6 Johannishof
7 Großer Stiftshof
8 Stiftskirche
9 Bibliothekstrakt (Krypta)
10 Marmortrakt (Sala terrena)
11 Innerer mittelalterlicher Klosterbereich (Kreuzgang)
12 Äußerer mittelalterlicher Klosterbereich

Figure 2. An example of an image which contains legend text, seen as rasterized text underneath the floorplan. Our legend key-grounding correctly detects the keys in the image and can successfully avoid incorrect grounding such as to the legend depicted below.

marking the keys of the legend in the image, mapping between the legend values and the key locations. The last subtask is obtained by searching for the keys in the image's OCR detected texts. Images can sometimes contain the full key–value legend itself in addition to the key markings (as seen in Figure 2 for example). To avoid marking these as well, we leverage the multiple text granularities returned by the Google Vision API and filter out identified keys that are part of a sentence/paragraph, and exclude areas detected as 'legend' by our layout component detector.

Architectural Information from the Image. As mentioned in the main paper, we use the OCR detections within the relevant layout components as candidates to include interesting architectural information – legends in `legend` areas, and architectural labels in `floorplan` areas. Next, we send these candidates to the LLM (similarly to the metadata legend extraction process) using the prompts in Figure 21 to obtain a raw legend/list of architectural labels. The legends are formatted and grounded similarly to those extracted from the metadata. The architectural features are translated if they are not in English using the Google Translate API while maintaining the original text representation which is used to ground them to the image.

B.5. Dataset statistics

Figure 5 shows a visualization of the different countries in our dataset; Figure 3 shows a histogram of common building types in the dataset; Figure 6 shows a visualization of the distribution of words in images detected using OCR; and Figure 4 show a histogram of the common architectural features that are grounded in *WAFFLE* images.

C. Experimental Details

C.1. Building Type Understanding Task

For the building type understanding task, we fine-tune CLIP on our training set. We train it for 5 epochs with batch size of 256, learning rate of 10^{-3} and Adam optimizer, on one A5000 GPU.

C.2. Open-Vocabulary Floorplan Segmentation Task

We learn image segmentation by fine-tuning CLIPSeg [6] (using base checkpoint `CIDAS/clipseg-rd64-refined`) on images with corresponding positive and negative segmentation maps.

Training details. Training data for segmentation is created from images with grounded architectural features (GAFs), using those that occur over 10 times in our dataset in order to filter out noise. To avoid leakage of information from rendered text, OCR detections are removed via in-painting as seen in Figure 7. During training, we also apply augmentations such as cropping, resizing and noising to enlarge our training dataset, applied to both images and target segmentation maps as needed.

To identify positive and negative targets for a given image and GAF text, we use the text embedding model `paraphrase-multilingual-mpnet-base-v2` from SentenceTranformers[*], measuring semantic similarity between GAF texts via embedding cosine similarity. Pairs of features with high (> 0.7) similarity scores are marked as positive and those with low (< 0.4) similarity are marked as negative; the loss is calculated on these areas alone.

Our overall loss consists of the weighted sum of three losses: cross-entropy over the masked positive and negative areas ($\mathcal{L}_{ce}$), L1 regularization loss ($\mathcal{L}_{L1}$) and entropy loss (mean of binary entropies of pixel intensities on the whole image) ($\mathcal{L}_e$). Our total loss is $\mathcal{L}_{total} = \frac{1}{2}\mathcal{L}_{ce} + \frac{1}{2}\mathcal{L}_{L1} + \mathcal{L}_e$.

We fine-tune with the following settings: 20 epochs; batch size 1; on one A5000 GPU; with a 10^{-4} learning rate; with an Adam [5] optimizer.

[*]https://www.sbert.net/

Evaluation. We manually annotate 95 images for evaluation with 27 common GAFs. The most common building types in our evaluation set are churches, castles and residential buildings.

C.3. Floorplan Generation Task

For the generation task, we adopt the text to image example provided in Hugging Face[*], by fine-tuning the Stable Diffusion (SD) [8] model `CompVis/stable-diffusion-v1-4`. We add a custom sampler to avoid over-sampling the same building; in particular, in each epoch we use only one sample out of all those corresponding to a given `<building_type>` and `<building_name>`. In addition, we resize the images to 512×512 keeping the original proportions of the image and adding padding as needed. We train our model for 20K iterations with batch size of 4, a learning rate of 10^{-5} and Adam optimizer, on one A5000 GPU.

For the boundary-conditioned generation task (conditioned on the outer contour of the building), we first extract the outer edges for all images in the training set. We use the Canny edge detection algorithm as implemented in the OpenCV library[*], extract only external contours[*], and remove contours with small areas. Samples where edge detection fails (returns an empty mask) are excluded. We then fine-tune[*] ControlNet [10]. We initialize the SD part of the architecture with our fine-tuned SD from the previous paragraph and the shape-condition part with a pre-trained model trained on Canny edges masks (as this condition is similar to our task) (`lllyasviel/sd-controlnet-canny`)[*] We use the same custom sampler and resize as described above. We train the model for 15K iterations with a batch size of 4, a learning rate of 10^{-5} and Adam optimizer, on one A5000 GPU.

For structure-conditioned generation (conditioned on building layouts), we use our fine-tuned SD model from above and fine-tune ControlNet on top of it. Unlike the boundary-conditioned task, we fine-tuned ControlNet using external data from the CubiCasa5K (CC5K) [4] train set. As conditions, we convert the structured CC5K SVG data into images with pixel values representing the subset of categories relevant to our dataset: foreground (white), background (black), walls (red), doors (blue), and windows (cyan). The foreground category comprises all CC5K room categories that are not background; doors and windows are

[*]https://huggingface.co/docs/diffusers/v0.18.2/en/training/text2image
[*]https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
[*]https://docs.opencv.org/4.x/d9/d8b/tutorial_py_contours_hierarchy.html
[*]https://huggingface.co/docs/diffusers/v0.18.2/en/training/controlnet
[*]https://huggingface.co/blog/controlnet

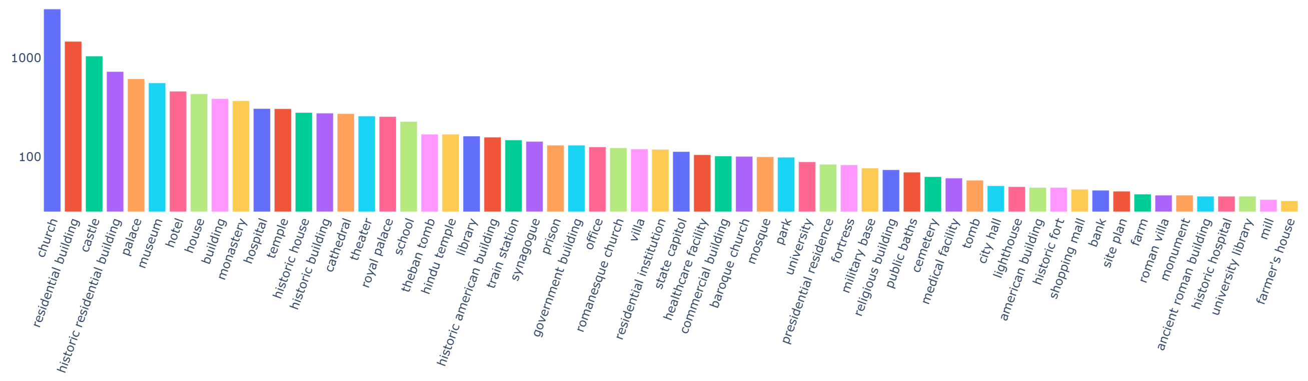


Figure 3. Distribution of common building types extracted automatically (log scale), illustrating the rich semantics captured in *WAFFLE*.

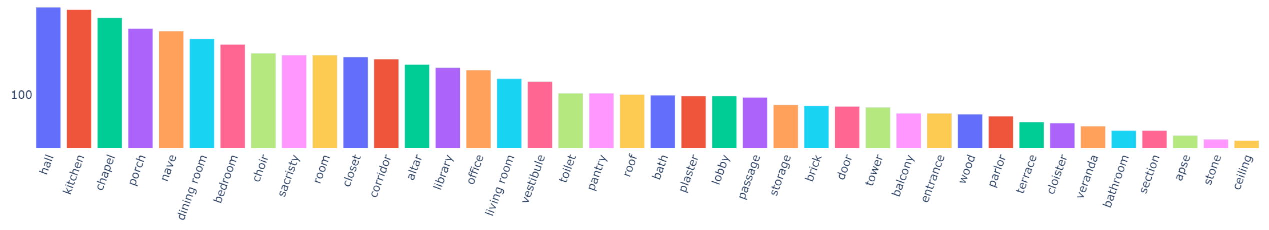


Figure 4. Distribution of the grounded architectural features (log scale), among almost 3K grounded images, 25K instances grounded, and 11K unique features.

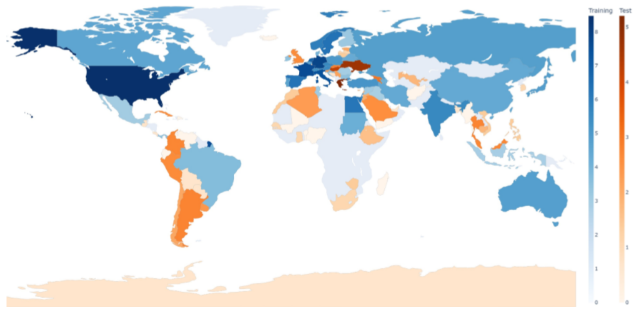


Figure 5. Number of samples per country (log scale) in *WAFFLE*, showing the diversity of our dataset for both training and test splits. Blue: training data; Orange: test data.

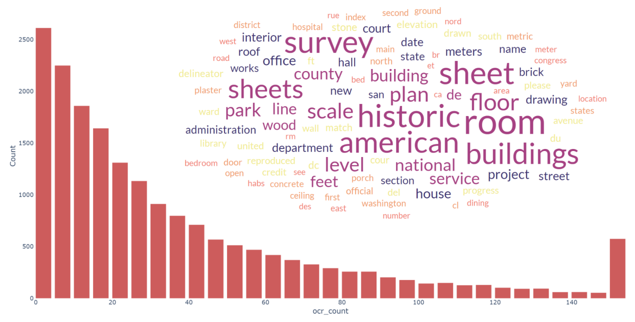


Figure 6. OCR words statistics. The bar plot depicts a histogram of the number of words detected in an image; the word map on the top right shows the most common words detected in our dataset. As illustrated above, the raw OCR data contains semantic information and also significant levels of noise, and thus it is challenging to operate over this data directly; hence motivating the need for extracting data from source external to the images (*e.g.* linked Wikipedia pages).

taken from the CC5K icon categories and overlaid on top of foreground/background/walls to produce the condition image (rather than being independent layers). We initialize the SD part of the architecture with our fine-tuned SD from the previous paragraph and the shape-condition part with the model's UNet weights. Images and conditions are resized using the method described above. We train the model for 20K iterations with a batch size of 4, a learning rate of 10^{-5} and Adam optimizer, on one A5000 GPU. During inference we use CFG scale 15.0 and condition scale 0.5, to fuse the style of the input prompt (learned from floorplans in *WAFFLE*) and the structure condition.

User study. Each study contained 36 randomly-generated

	Building	Castle	Cathedral	Church	Historic building	Hospital	Hotel	House	Monastery	Museum	Palace	Residential building	Temple	School	Theater
FID ↓															
SD	284.6	159.6	198.4	188.3	285.1	199.2	194.3	212.2	159.2	180.6	139.6	224.0	**165.9**	141.6	**189.5**
SD$_{FT}$	**148.3**	**146.1**	**156.8**	**142.0**	**147.7**	**114.5**	**100.9**	**158.0**	**122.3**	**137.6**	**119.1**	**168.6**	176.9	**102.0**	238.4
KMMD ↓															
SD	0.13	0.06	0.11	0.09	0.16	0.09	0.13	0.12	0.08	0.07	0.05	0.13	**0.07**	0.06	**0.08**
SD$_{FT}$	**0.07**	**0.05**	**0.06**	**0.04**	**0.11**	**0.05**	**0.05**	**0.08**	**0.03**	**0.05**	**0.04**	**0.11**	0.09	**0.03**	0.17
CLIP Sim. ↑															
SD	25.3	25.2	24.4	24.2	**25.3**	24.0	24.2	25.8	25.9	25.2	**25.5**	25.1	24.7	24.6	**24.6**
SD$_{FT}$	**25.9**	**25.6**	**25.6**	**25.4**	**25.3**	**24.5**	**25.1**	**26.7**	**26.5**	**26.1**	**25.5**	**26.0**	**25.6**	**25.8**	24.5

Table 1. Quantitative results of results on floorplan image generation split by building type, comparing the quality of images generated with the pretrained model and our fine-tuned model.

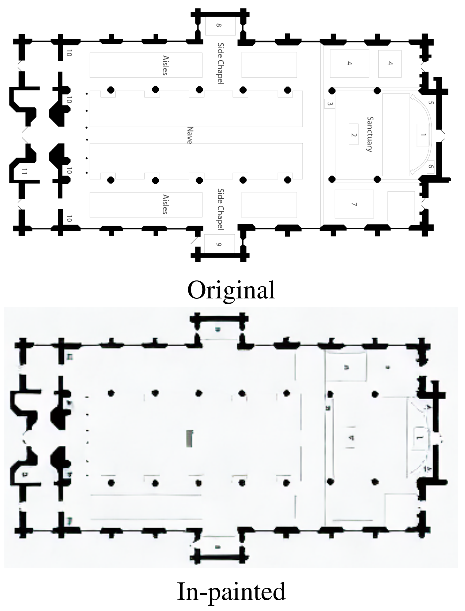


Figure 7. An example of in-painting to preprocess data for the segmentation task. On the left, we show the original image which contains text indicating architectural features, including the *Nave*, *Side Chapels* and *Aisles*. On the right, we show the in-painted version of the image, which succeeds in removing these texts to prevent leakage. We observe that this in-painting process may slightly modify the appearance of the image, but the floorplan's structure is mostly preserved.

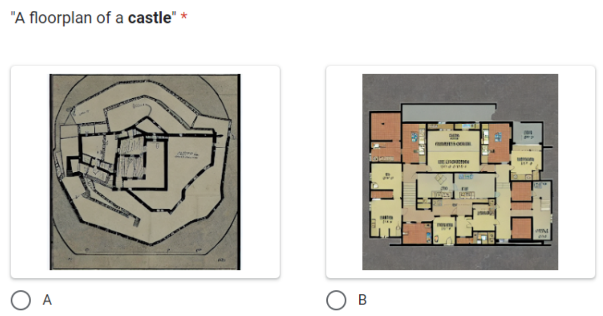


Figure 8. A sample question from our user study on text-conditioned floorplan generation.

image pairs, with text prompts mentioning various building types that were sampled from the 100 most common types. Overall, thirty one users participated in the study, resulting in a total of 1,116 image pairs (one generated from the pretrained model, and the other generated from the finetuned model) that were averaged for obtaining the final results reported in the main paper.

Participants were provided with the following instructions: *In this user study you will be presented with a series of pairs of images, generated by the prompt: "a floorplan of a <BUILDING_TYPE>". For example, "a floorplan of a cathedral". For each pair, please select the image that best conveys the text prompt (i.e., both looks like a **floorplan diagram**, and also looks like a plan of the specific **building type** mentioned in the prompt). If you are unsure, please make an educated guess to the best of your ability. Thank you for participating!*

A sample question from our user study is illustrated in Figure 8. All of the questions were forced-choice, and participants could only submit after answering all of of the

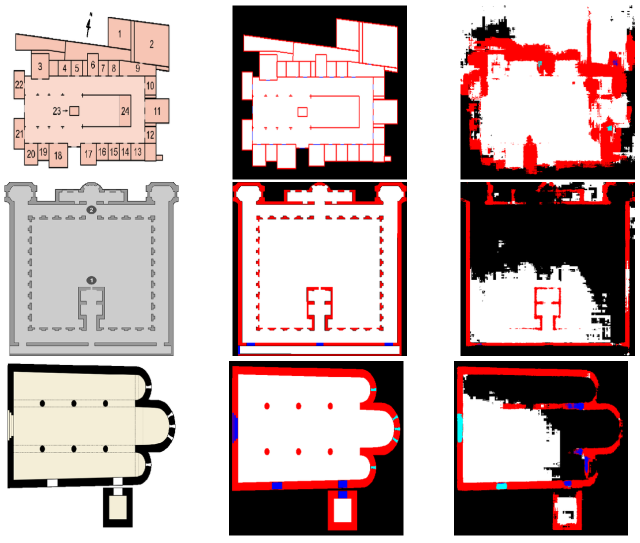


Figure 9. Benchmark for semantic segmentation (over the walls, doors, windows, interior and background categories) on images from *WAFFLE* using the strong supervised CC5K [4] pretrained model. We can see that our data serves as a challenging benchmark as the model struggles with more diverse and complex floorplans.

questions.

C.4. Benchmark for Semantic Segmentation

We created a benchmark of 110 SVG images, containing wall, windows and door annotations. We included SVG images from our test set. To obtain additional SVG images, we also searched for SVGs that were filtered during the dataset filtering step. Then, we used Inkscape [*] which allowed us to easily annotate full SVG components at once instead of doing it pixel-wise. This made the manual annotation process less tedious and more accurate.

C.5. Wall Segmentation with a Diffusion Model

We apply a diffusion-based architecture to wall segmentation by training ControlNet [10] using CubiCasa5K [4] (CC5K) layout maps as the target image to generate and input images as conditions. In particular, we convert CC5K annotations into binary images by denoting walls with black pixels and use these as supervision for binary wall segmentation. We initialize with the `CompVis/stable-diffusion-v1-4` checkpoint and train for 200K iterations on train items in the CC5K dataset which provides us with 4,200 pairs of images. Other training hyperparameters are the same as those used for ControlNet applied to other tasks as described above. During inference, we input an image (resized to the correct resolution) as a control, generate 25 images with random seeds (and guidance scale 1.0, CFG

[*]https://inkscape.org/

	ResNet	Diffusion
Precision	0.737	**0.746**
Recall	0.590	**0.805**
IoU	0.488	**0.632**

Table 2. A comparison between an existing ResNet-based wall detection model (introduced in CC5K [4]) and a Diffusion-model based one (detailed further in Section C.5), evaluated on our benchmark. We can see the Diffusion-based model outperforms the ResNet-based model across all metrics, suggesting that newer architectures show promise in improving localized knowledge of in-the-wild data, such the floorplans found in *WAFFLE*.

scale 7.5, 20 inference steps using the default PNDM scheduler). We discretize output pixel values to the closest valid layout color and then use pixel-wise mode values, thus reducing noise from the random nature of each individual generation.

D. Additional Results and Visualizations

D.1. Semantic Segmentation Results

Figures 9 and 10 contain examples of test images and annotations from our benchmark for semantic segmentation, and the results of the existing CC5K [4] model on them. These figures demonstrate how the model is challenged in detecting segments it wasn't exposed to during training, like pillars or curved walls. In Figure 11 we show qualitative examples of wall detection using different model architectures – the existing ResNet-152 [3] based model, and the Diffusion-based model discussed in Section C.5. As illustrated in the figure, using a more advanced model architecture allows for obtaining significantly cleaner wall segmentations. Table 2 contains a quantitative analysis of the two models on the wall segmentation prediction task.

D.2. Additional Open-Vocabulary Floorplan Segmentation Results

In Figure 12, we show additional examples of text-driven floorplan image segmentation before and after fine-tuning on our data. We see that the baseline model struggles to localize concepts inside floorplan images while our fine-tuning better concentrates probabilities in relevant regions, approaching the GT regions indicated in orange rectangles.

In Figure 13 we visually compare the segmentation results to those of CC5K and CLIPSeg on residential buildings. We observe that the supervised CC5K model (trained on Finnish residential floorplans alone) fails to generalize to the diverse image appearances and building styles in *WAFFLE*, even when they are residential buildings, while our model shows a more general understanding of semantics in such images.

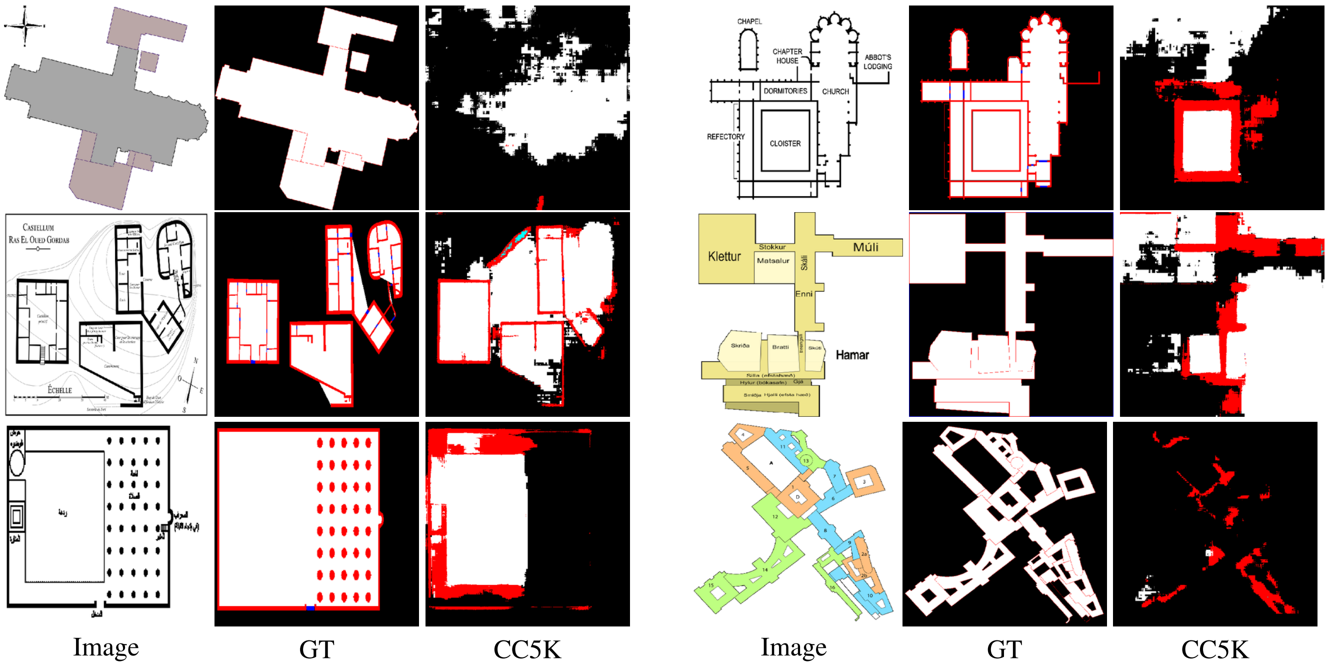


Figure 10. Additional results on the semantic segmentation benchmark. Images are annotated according to walls, doors, windows, interior and background categories. On the right we show results obtained with CC5K [4].

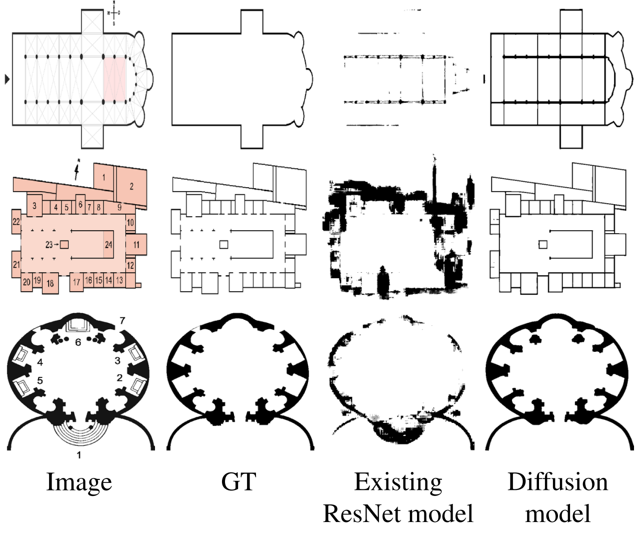


Figure 11. Wall segmentation results, comparing the CubiCasa5K (CC5K) [4] baseline segmentation model that uses a ResNet backbone to a modified architecture that uses a Diffusion model, as described in Section C.5. The Diffusion-based model yields refined wall predictions, as illustrated by the examples shown above.

D.3. Additional Generation Results

We show additional results for the generation task in Figure 14 and for the spatially-conditioned generation in Figure 15. We provide multiple examples for various building types, showing that a model trained on our data learns the distinctive structural details of each building type. For example, castles have towers, libraries have long aisles, museums, hospitals, and hotels have numerous small rooms, churches have a typical cross shape, and theaters are characterized by having rows of seats facing a stage. The differences between the various types and their unique details are further shown in Figure 17, where we illustrate examples from our training set of various types.

In Table 1 we show a breakdown of the metrics for the generated images according to the most common building types in the dataset. The table compares our fine-tuned model with a base SD model, showing that for the vast majority of building types, our fine-tuned model generates images that are both more realistic and also semantically closer to the target prompt.

For structure-conditioned generation, we show additional results in Figure 16, where input conditions are derived from the CC5K dataset annotations as described above. In the figure, we show the effect of changing the condition and CFG scales during inference, illustrating the significance of these settings. In particular, we see that the condition scale controls the trade-off between fidelity to the layout condition and matching the building type in the prompt (rather than exclusively outputting images in the style of the CC5K fine-tuning data).

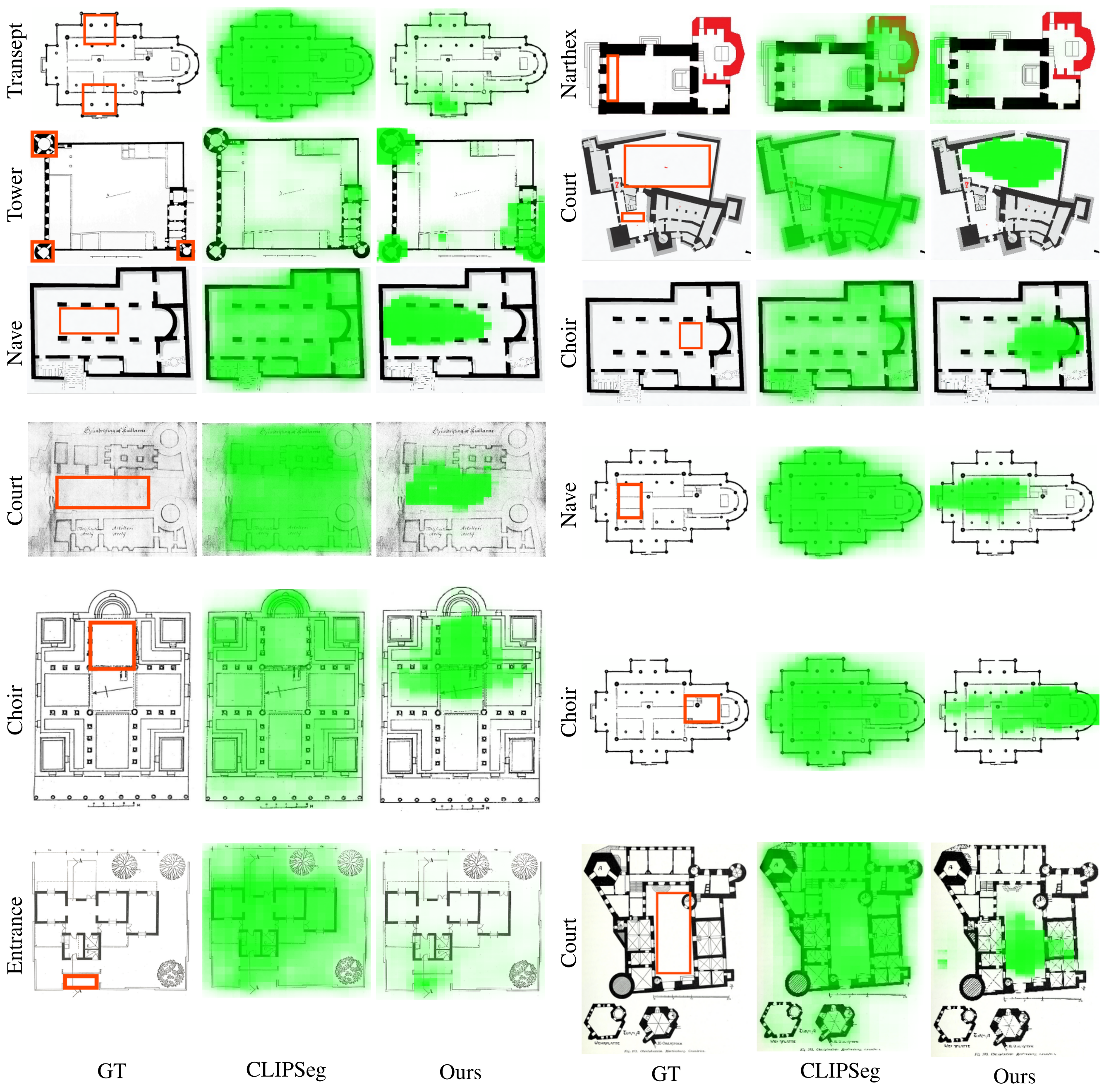


Figure 12. In each column: segmentation results on samples of our test set before (center) and after (right) fine-tuning on our data.

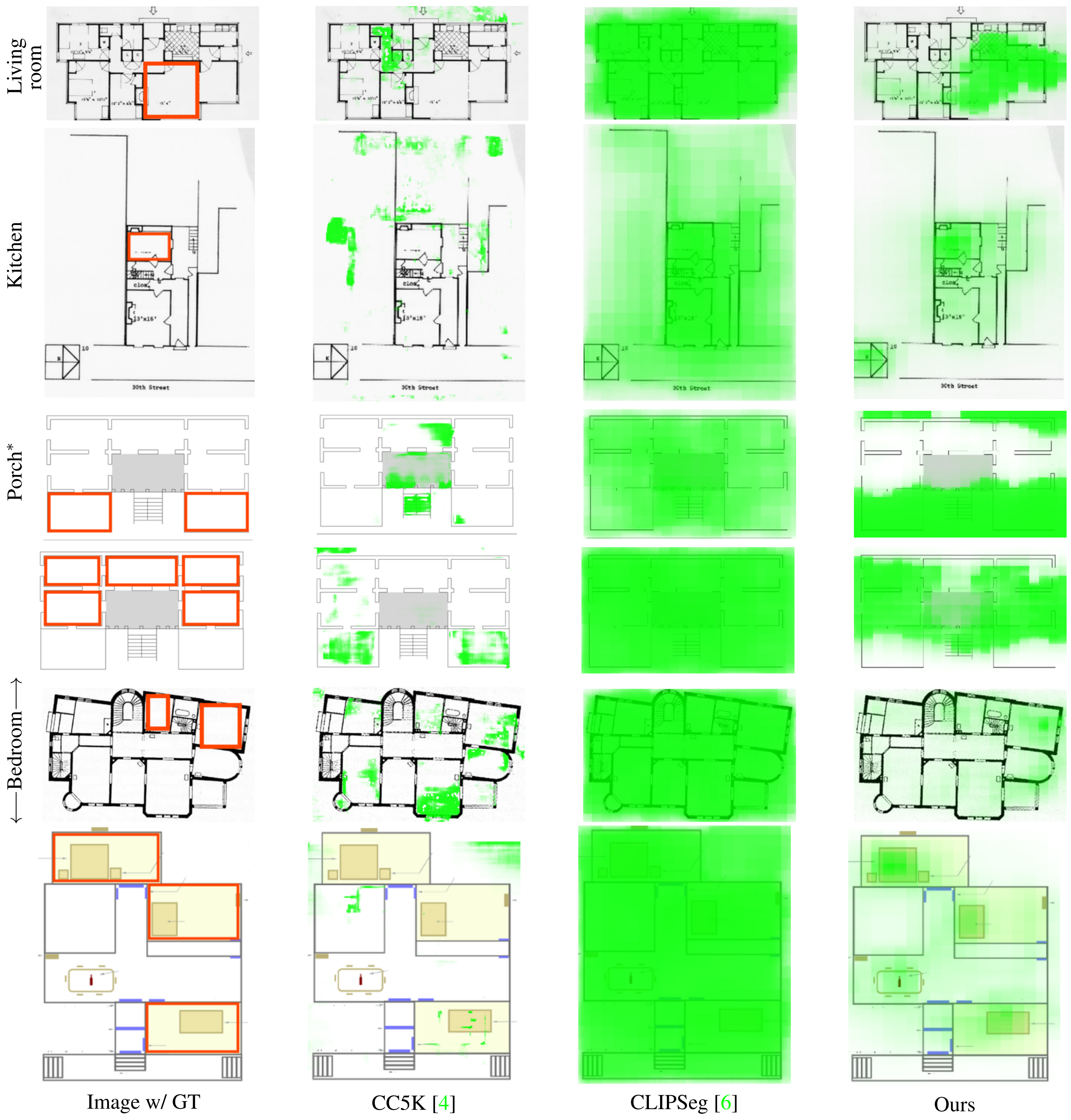
References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 1

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 2

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 6

[4] Ahti Kalervo, Juha Ylioinas, Markus Häikiö, Antti Karhu, and Juho Kannala. Cubicasa5k: A dataset and an improved multi-task model for floorplan image analysis. In *Image Analysis: 21st Scandinavian Conference, SCIA 2019,*



*Our *porch* results correspond to the CubiCasa5K *outdoor* category.

Figure 13. Additional comparisons of our segmentation probability map results on residential buildings with the strongly-supervised CubiCasa5K (CC5K) model [4].

Norrköping, Sweden, June 11–13, 2019, Proceedings 21, pages 28–40. Springer, 2019. 3, 6, 7, 9

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 3

[6] Timo Lüddecke and Alexander Ecker. Image segmentation using text and image prompts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7086–7096, 2022. 3, 9

[7] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya

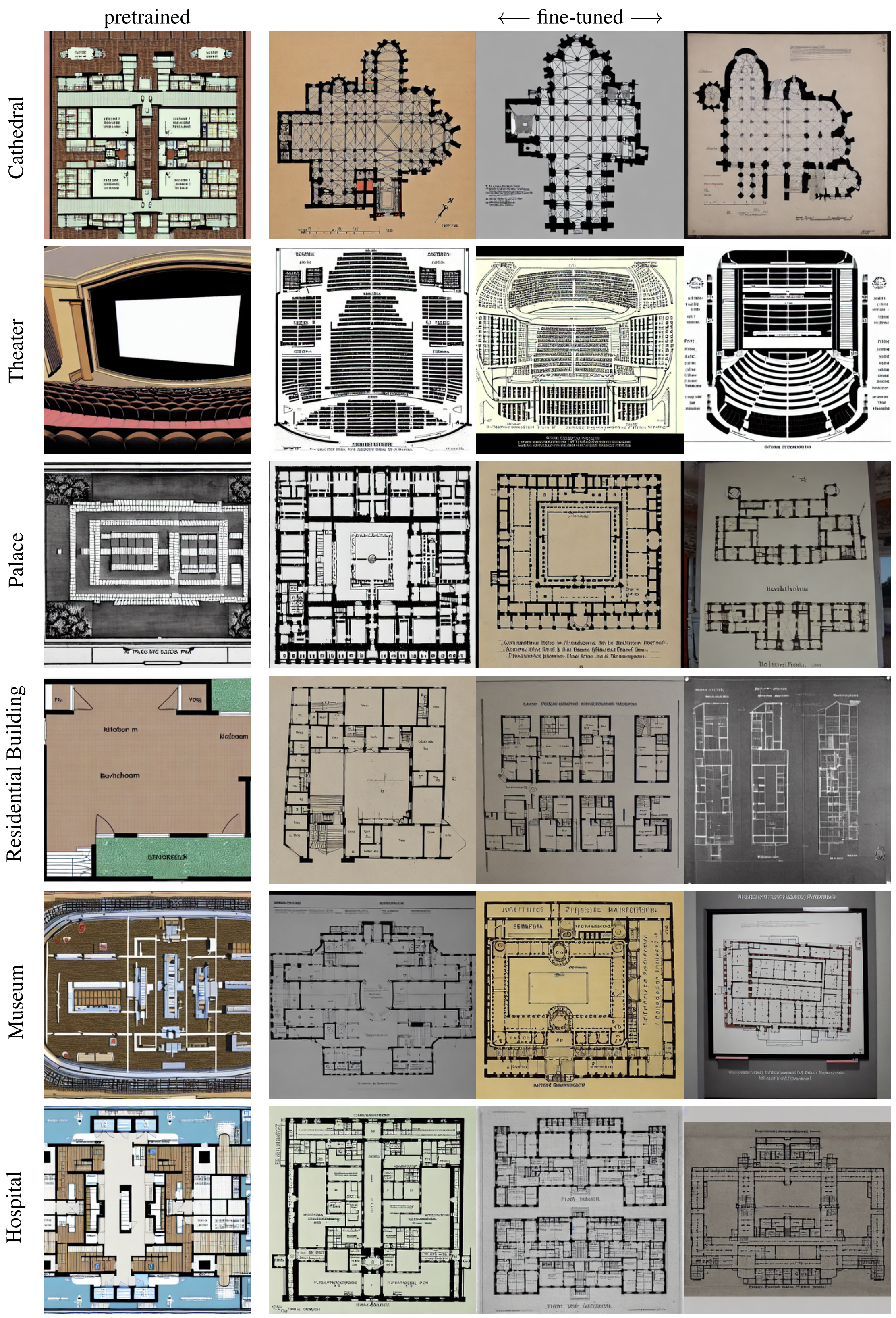


Figure 14. Additional generated floorplans, showing diverse building types (provided on the left). The left-most column shows samples from the pretrained SD model and rest of the columns showcase the results from our fine-tuned model.

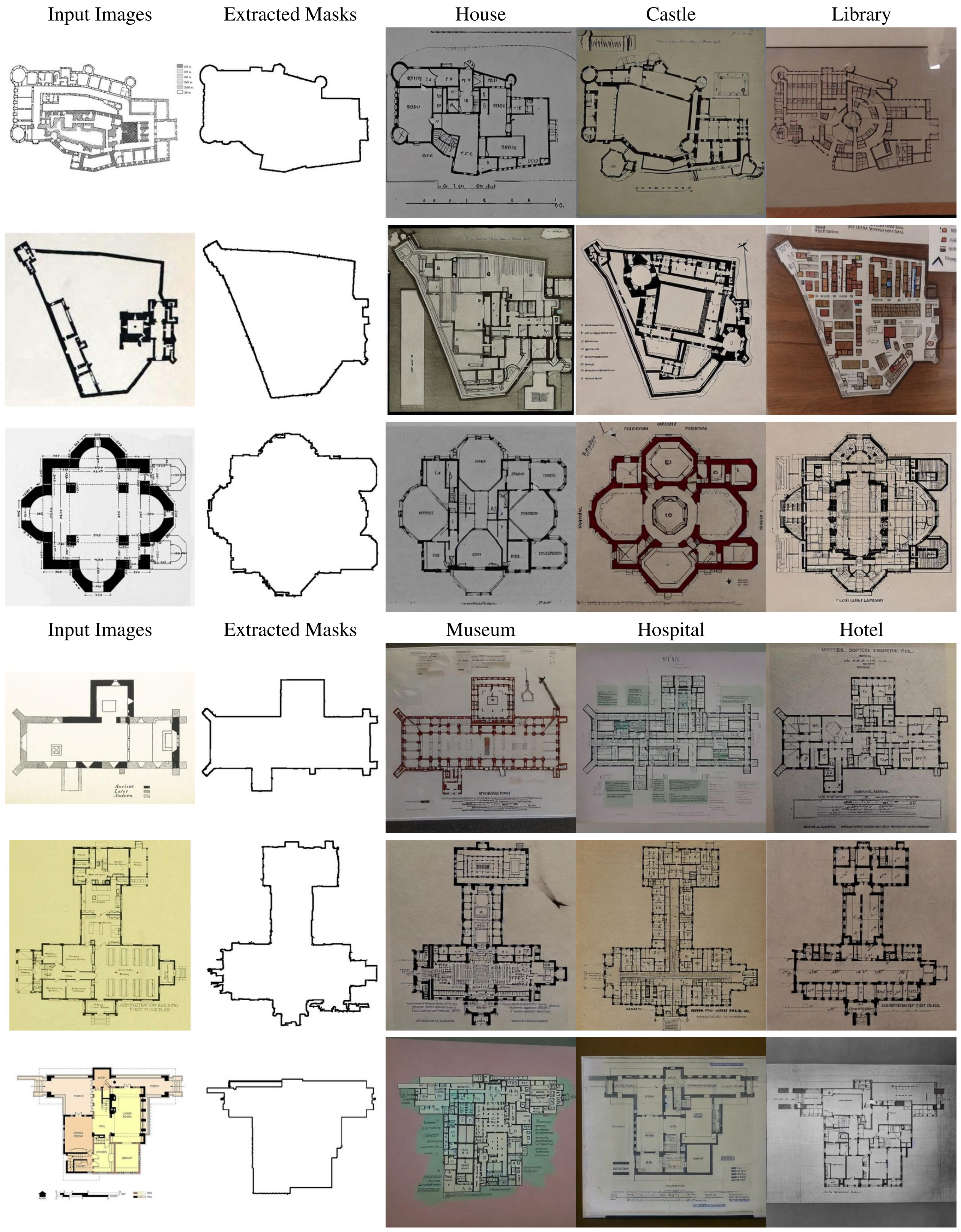


Figure 15. Additional results for boundary-conditioned generation, showing a variety of shapes (shown on the left) and building types (shown on top).

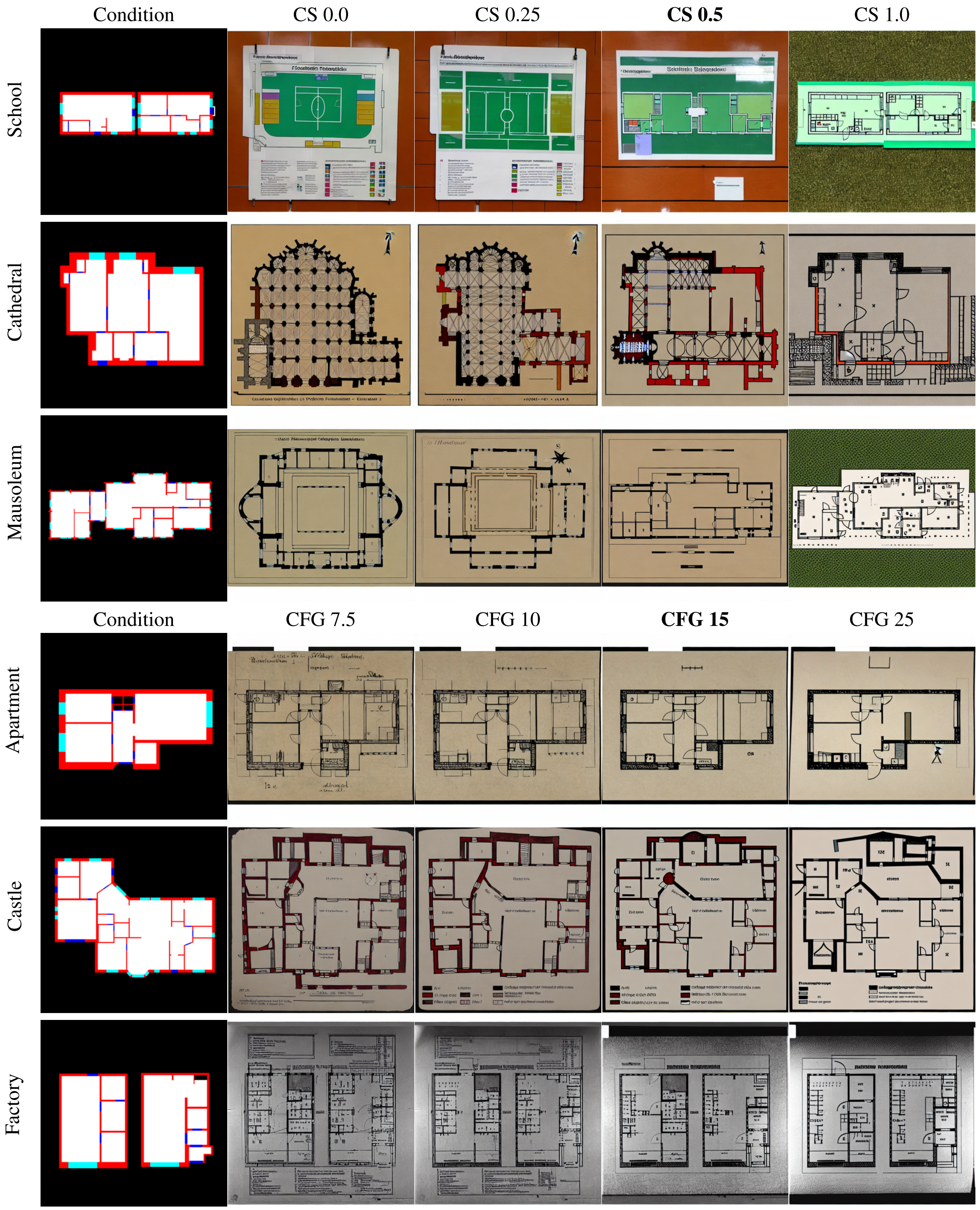


Figure 16. Additional results for structure-conditioned generation, showing the effect of changing condition scale (CS) and CFG scales during inference (with a fixed seed). The condition scale controls the trade-off between adherence to the structure condition and avoiding leakage of the CubiCasa5K style which ControlNet was exposed to in fine-tuning. We also find a relatively high CFG value to improve image quality. Chosen values for inference are in **bold**.

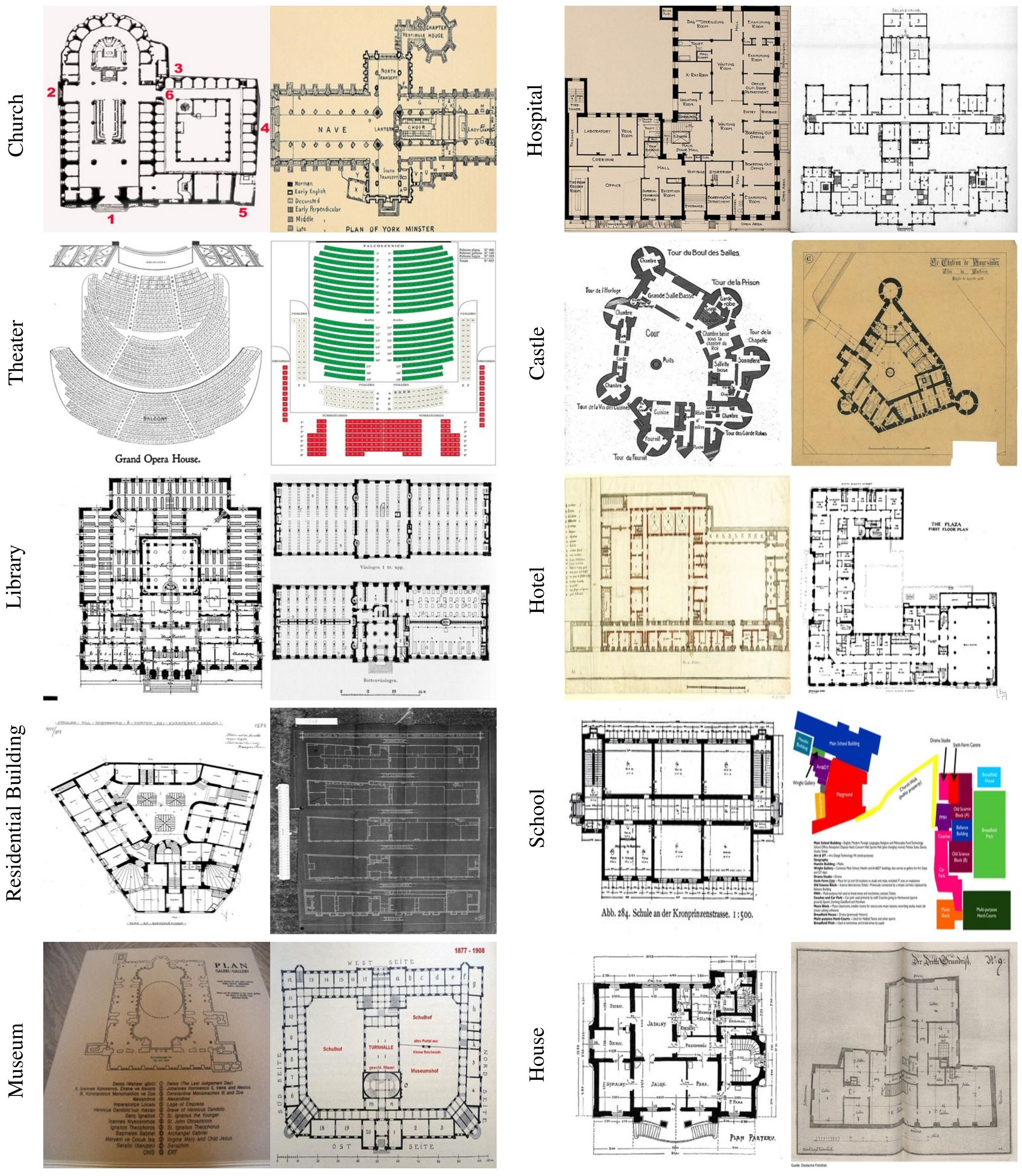


Figure 17. Examples of images from our dataset with their building types (shown on the left)

Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 1

[8] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2022. 3

[9] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov,

Image Category

```
[INST]

Please read the following
(truncated) information
extracted from Wikipedia
related to an image:

--- START WIKI INFO ---

* Entity category:
  {category}
* Entity description:
  {description}
* Image filename:
  {fn}
* Texts that appear in the
  (image extracted with OCR)
  {ocr_texts}

--- END WIKI INFO ---

Now answer the following
question in English:  What is
this file most likely a
depiction of?
(A) A floorplan
(B) A building
(C) A cross section of a
building
(D) A garden/park
(E) A Map
(F) A city/town
(G) A physics/mathematics topic
(H) I don't know

Please choose one answer
(A/B/C/D/E/F/G/H)

[/INST]

The best answer is (
```

Building Name

```
[INST]

Please read the following
(truncated) information
extracted from Wikipedia
related to an image of a
building:

--- START WIKI INFO ---

* Entity category:
  {category}
* Entity description:
  {description}
* Image filename:
  {fn}
* Wiki page summary:
  {wiki_shows}
* Texts that appear in the
  image (extracted with OCR):
  {ocr_texts}

--- END WIKI INFO ---

What is the name of the
building depicted above?
Write it in English,
surrounded by brackets < >

[/INST]

The name of the building
discussed by the article is <
```

Building Type

```
[INST]

Please read the following
(truncated) information
extracted from Wikipedia
related to an image of the
building {building_name}:

--- START WIKI INFO ---

* Entity category:
  {category}
* Entity description:
  {description}
* Image filename:
  {fn}
* Wiki page summary:
  {wiki_shows}

--- END WIKI INFO ---

What type or category of
building is {building_name}?
Write your answer in
English, surrounded by
brackets < >

[/INST]

The building {building_name}
is a <
```

Location Information

```
[INST]

Please read the following
(truncated) information
extracted from Wikipedia
related to an image of the
building {building_name}:

--- START WIKI INFO ---

* Entity category:
  {category}
* Entity description:
  {description}
* Image filename:
  {fn}
* Wiki page summary:
  {wiki_shows}

--- END WIKI INFO ---

Where is {building_name}
located?  Write the country,
state (if exists) and city
surrounded by brackets < >
and separate between them
with a semi colon, for example:
<City; State; Country>.
If one of them is unknown
write 'Unknown', for example:
<City; Unknown; Country>,
<Unknown; State; Country>
etc.

[/INST]

{building_name} is located in <
```

Figure 18. The prompts used for LLM-based extraction of pGTs. Each {...} placeholder is replaced with the respective image data. At first we only have raw data (as seen in the "Image Category" prompt), but once we gather pGTs we may use them in other prompts, for example {*building_name*} as used in "Building Type" and "Location Information". We ask the LLM to return a semi-structured response (choosing an answer from a closed set; wrapping the answer in brackets etc.) so that we can easily extract the answer of interest. From left to right: The "Image Category" prompt is used for the initial text based filtering, where categories (A) and (B) are positive and the rest are negative. The "Building Name" and "Building Type" prompts are used for setting the building name and type respectively. The "Location Information" prompt extracts the country, state, and/or city (whichever of these exist). Note that the country is subsequently used for defining our test-train split.

Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 1

[10] Lvmin Zhang, Anyi Rao, and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023. 3, 6

Legend Existence (Wikipedia)

```
[INST]

The image "{fn}" is a plan of
the building {building_name} and
it contains the following texts,
detected by an OCR model:

{ocr_texts}

Please read the following
excerpt from an article about
the building which contains
this image:

--- START EXCERPT ---

{snippet}

--- END EXCERPT ---

Now answer the following
question about the excerpt:
Does the excerpt contain a
legend for the image "{fn}",
i.e.  an itemized list
corresponding to regions marked
by OCR labels in the image,
explaining what each label
signifies?  Answer
yes/no/unsure.

[/INST]

The answer to the question is:
```

Legend Content (Wikipedia)

```
[INST]

The image "{fn}" is a plan of
the building {building_name} and
it contains the following texts,
detected by an OCR model:

{ocr_texts}

Please read the following
excerpt from an article about
the building which contains
this image:

--- START EXCERPT ---

{snippet}

--- END EXCERPT ---

The excerpt contains a legend,
i.e.  an itemized list
corresponding to regions
marked by labels in the image.
Reproduce the legend below.

[/INST]

Sure!  Here is the legend:
```

Legend Existence (caption)

```
[INST]

The image "{fn}" is a plan
of the building {building_name}
and it has the following
description:

--- START IMAGE DESC. ---

{description}

--- END IMAGE DESC. ---

Does the description above look
like it contains a legend for
the image, i.e.  an itemized
list corresponding to regions
marked by labels in the image,
explaining what each label
signifies?
Write yes/no/not sure in
English, surrounded by
brackets < >

[/INST]

<
```

Legend Content (caption)

```
[INST]

The image "{fn}" is a plan
of the building {building_name}
and it has the following
description:

--- START IMAGE DESC. ---

{description}

--- END IMAGE DESC. ---

Does the discussed image
contain a legend (as in a
key/table/code for
understanding the image)?
If so, what are the legend's
contents?  Answer with a
bulleted list in English of the
legend contents.  Include only
full items and not just labels
(for example, '1.  nave' should
be included, but '1.'  alone
shouldn't)

[/INST]

Answer:  The legend contains:
*
```

Figure 19. The prompts used for extracting the legend contents. The two left prompts are used for extracting data from Wikipedia, and the two right ones for the image caption. In both cases this is a two-step extraction: first we ask the LLM if it thinks the text contains a legend. Only if it answers yes, we ask it for its content. This reduces hallucinations and keeps the answers accurate.

Legend simplification

```
[INST]

The following texts contain a legend of a {building_type} floor
plan in a key:value format:

--- START LEGEND ---

{legend}

--- END LEGEND ---

Please rewrite the legend using simple and generic words.

Do:
* Include all legend parts from the list above.
* Keep it simple and short:  summarize each row in one/two words
* Keep the original legend keys
* In case the features have distinct names (e.g.  Chapel of the
Ascension) treat their type only (e.g.  a chapel) and disregard
any specific name.

Don't:
* Don't invent new information
* Don't include specific names (use their type instead)
* Don't skip any of the legend lines above

Write your answer in English, translating any non-English terms.

[/INST]

Sure, here's a simplified and generalized version of the legend:
*
```

Figure 20. The prompt used for legend simplification, serving to clean up the original legends for the image grounding process. The goal is to obtain a list of keys to architectural features. We aim to shorten long descriptions, remove names, and translate any non-English text.

Legend Existence	Legend Content	Arc-Feats Existence	Arc-Feats Content
[INST] The image "{fn}" is a plan of the building {*building_name*} and it contains the following texts, detected by an OCR model: --- START IMAGE TEXTS --- {*ocr_legend_candidate*} --- END IMAGE TEXTS --- Do the OCR detections above look like they contain a legend for the image, i.e. an itemized list corresponding to regions marked by OCR labels in the image, explaining what each label signifies? Write yes/no/not sure in English, surrounded by brackets < > [/INST] **<**	[INST] The image "{fn}" is a plan of the building {*building_name*} and it contains the following texts, detected by an OCR model: --- START IMAGE TEXTS --- {*ocr_legend_candidate*} --- END IMAGE TEXTS --- The above texts may contain, among other things, the content of an image legend (as in a key/table/code for understanding the image). Can you extract the legend contents from the above texts? Answer with a bulleted list of the legend contents. Include only full items and not just keys/labels (for example, '1. nave' can be included, but '1.' or 'nave' alone shouldn't). Disregard text that doesn't seem like it's part of the legend. Include the original keys/labels and don't invent new ones. If you can't deduce a legend return "I don't know". [/INST] **Sure! Here are the legend contents:**	[INST] The image "{fn}" is a plan of the building {*building_name*} and it contains the following texts, detected by an OCR model: --- START IMAGE TEXTS --- {*ocr_legend_candidate*} --- END IMAGE TEXTS --- Do the OCR detections above look like they contain words that represent architectural feature labels? Disregard anything that looks like a symbol or a key (like numbers), and any words that represent direction (e.g. north, east, etc.) Write yes/no/not sure in English, surrounded by brackets < > [/INST] **<**	[INST] The image "{fn}" is a plan of the building {*building_name*} and it contains the following texts, detected by an OCR model: --- START IMAGE TEXTS --- {*ocr_legend_candidate*} --- END IMAGE TEXTS --- The above texts may contain, among other things, architectural features marked on the floorplan. Out of the texts above, can you extract those that represent architectural features? Like room types, halls, porches, etc. Don't include anything that looks like a symbol or a key (like numbers). Don't include any words that represent direction (e.g. north, east, etc.). Disregard text that isn't related to architectural features. Answer with a bulleted list of the architectural features. Use the original text, do not modify, translate, or add extensions to the text you chose to add. If you can't deduce any architectural features return "I don't know". [/INST] **Sure! Here are the architectural features that appear in the texts you provided:**

Figure 21. The prompts used for extracting legends and architectural features from OCR detections. The two left prompts are used for extracting legends, and the two right ones for architectural feature labels. In both cases this is a two-step extraction: first we ask the LLM if it thinks the text contains a legend. Only if it answers yes, we ask it for its content. This reduces hallucinations and keeps the answers accurate.