Supplementary Material/Appendix

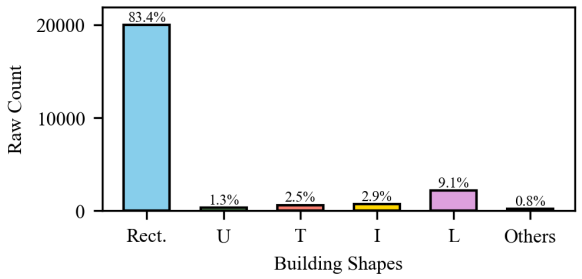

Figure A1: Histogram of dataset building shapes.

A6 Additional Quantitative Motivations and Qualitative Results

This section provides an expanded discussion and more results of our method.

Robustness using RUTIL. Our shape types i.e., rectangle, U-shaped, T-shaped, I-shaped, and L-shaped (RUTIL) builds upon prior CV works (Castagno and Atkins 2018; Steadman 2006; Zeng et al. 2018). The vast majority of building polygons can be represented with RUTIL (99%). Please see Fig. A1. For exceptions, we best fit a shape by optimizing for maximum IoU and mininum area difference.

Performance on INRIA Benchmark Dataset. In addition to the prior comparison to SOTA methods, we show that our method performs well on structured urban areas, of which INRIA (Zhang et al. 2022) is a benchmark. Our method reachesan IoU of 0.96 of INRIA benchmark GT (see Fig. A3 for performance in Chicago and Austin).

Building Density vs Pixel Size. We analyze the performance of our method, and compare to baselines, for various building densities and pixel sizes. Building densities range from just 20 buildings in the image up to 500 buildings in the image. Pixel size ranges from 0.5 meters per pixel (mpp) to 120 mpp, encompassing various standard satellite resolutions such as Planet Dove (PlanetLabs 2024a) and NASA Landsat (NASA 2024), to name a few. The surface plot in Fig. A2 illustrates the robustness and performance of our method across a multitude of pixel sizes and densities. As expected a high density of buildings and a big pixel size (e.g., 300-500 buildings per image and 120 mpp) performs the worst. However, as the pixel size reduces, the building density reduces, or both performance in general goes up and ours always outperforms all baselines. In our paper, we typically tackle building density of 300-500 and 0.5 mpp.

Additional Counting Comparison Results. In the main paper, we visualized the comparative state of our model being faster than all models with $AP > 70$ while being over 38% more effcient than the next-best model. In Tab. A1, we present the raw counting performance in a comparative setting.

Additional Center Extractor Comparison Experiment. We report the comparison of segmentation performance (AP), count accuracy (Acc.), and efficiency across our center extractor and SOTA tracking models in Tab. A2. We find our AP, count accuracy, and efficiency to be approximately 9%, 11%, and 34% higher than the next best center extractor used. Futhermore, since this phase comes as the first step of the pipeline, we show how sensitive perturbation of centers in this step. We observe very low sensitivity which is desirable and robust (see Tab. A2).

Resolution-varied Cross Validation across Satellite Types. We perform a resolution-based cross-validation across standardized satellite resolutions is presented in Tab. A3 in Appendix. We observe high accuracy in SkySat 0.5 mpp (PlanetLabs 2024b) and Dove Satellite 3 mpp (PlanetLabs 2024a) and a minor dip in NASA LandSat (NASA 2024). When we train with all resolutions combined, it performs above $AP = 77$ in all cases.

Additional Qualitative Results. In the main paper, we presented pictorial comparisons to three baselines: LOG-CAN++ (Ma et al. 2025), BoundaryFormer (Lazarow, Xu, and Tu 2022), and U2Seg (Niu et al. 2024). In Fig. A5, we illustrate the qualitative comparisons to the remainder of the baselines: SAM 2 (Ravi et al. 2025), DiffusionDet (Liu, Ren, and Zhao 2023), Mask2Former (Cheng et al. 2022), and Mask Scoring RCNN (MS-RCNN) (Huang et al. 2019).

Rationale for Patch Size. As part of our node features, we used image patches (approximately) capturing each building. To find the best patch size, we experimented with 5 meter (m) increments in both height and width up to 30m $\times$ 30m. Our objective was to roughly capture the boundary of one building on average. Besides plotting AP, AP_{50}, and AP_{75}, we also plotted the count accuracy ($Acc.$). Intuitively, in Fig. A4 and Tab. A4, we found the peak AP and peak $Acc.$ at the same patch size of 27m $\times$ 27m. This image patch size is used in our provided results.

A7 Additional Loss Function Experiment

All our geometry and patch features are continuous floating points (normalized). Only the valid flag, and the shape type features are two discrete features. We achieved convergence using the unified loss function presented in the main paper in Eq. 8. Additionally, we experimented with cross entropy loss function as:

$$\mathcal{L}' = -\frac{1}{|\mathcal{V}|} \sum_{v \in \mathcal{V}} \mathbf{w} \odot \sum_{c=1}^{C} \mathbf{x}_v^{(c)} \log \mathbf{z}_v^{(c)} \tag{9}$$

where the cross-entropy loss $\mathcal{L}'$ is computed over a set $\mathcal{V}$ of vertices, where $|\mathcal{V}|$ denotes the total number of nodes. For each node $v \in \mathcal{V}$, the GT shape type discrete label is represented by $\mathbf{x}_v$, which is a one-hot encoded vector. The model's predicted probability distribution $\mathbf{z}_v$ over C discrete classes $C = 5$ for building shape types - R, U, T, I, L), where $\mathbf{z}_v^{(c)}$ represents the probability assigned to class c. The loss is computed as the weighted sum of the negative log-likelihood of the predicted probabilities, where $\mathbf{w}$ is our element-wise weighting factor (tuned through training), and $\odot$ denotes element-wise multiplication.

We find that both $\mathcal{L}$ in Eq. 8 and $\mathcal{L}'$ in Eq. 9 achieved convergence. For the former, we only need to optimize the equation for $\mathcal{L}$ over all the output features. However, for the

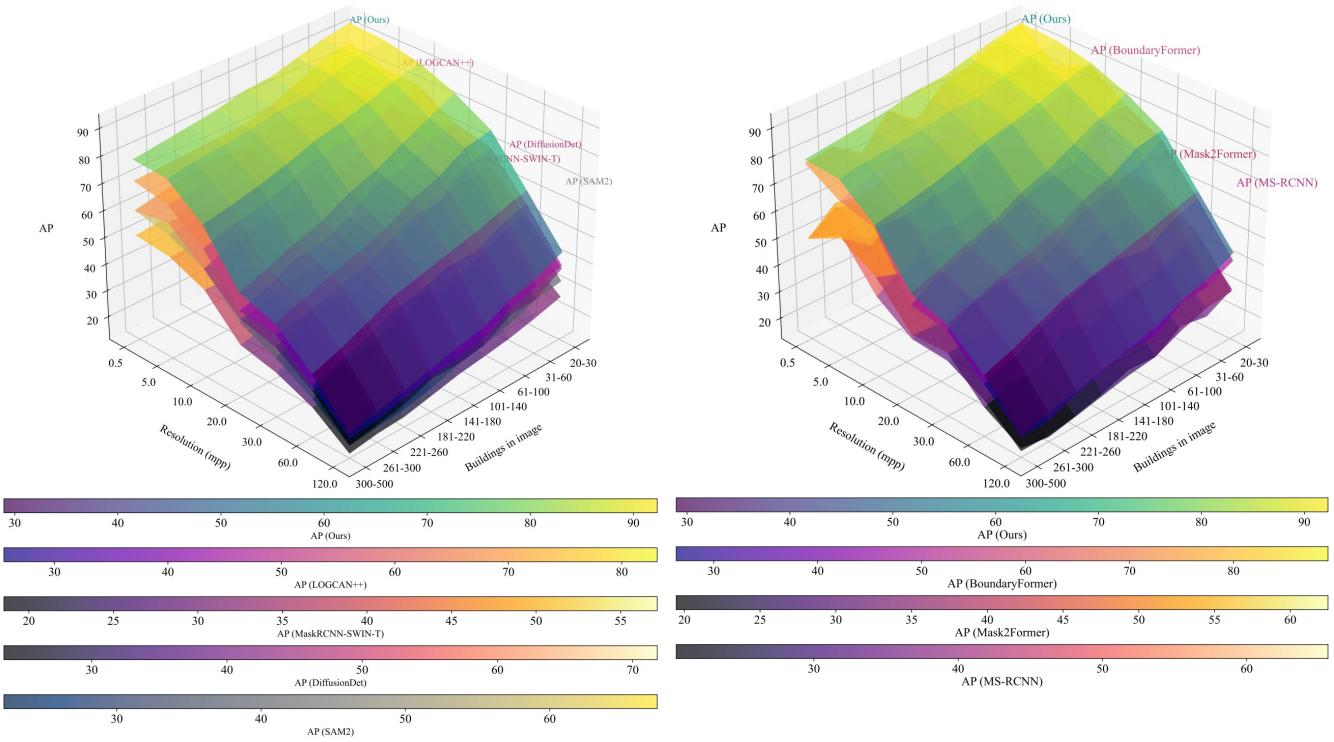

Figure A2: **Comparative Performance varying Density and Pixel Size.** Our segmentation performance (AP) outperforms all baselines, on all levels of density and by larger margin on denser cases. Further, the performance remains robust across various pixel sizes.

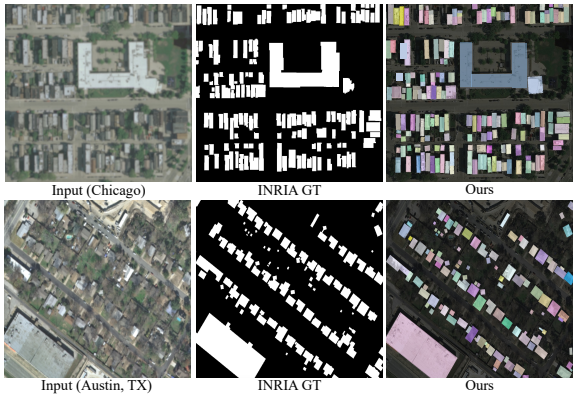

Figure A3: **Benchmark Dataset Performance.** Comparison with benchmark INRIA Ground Truth.

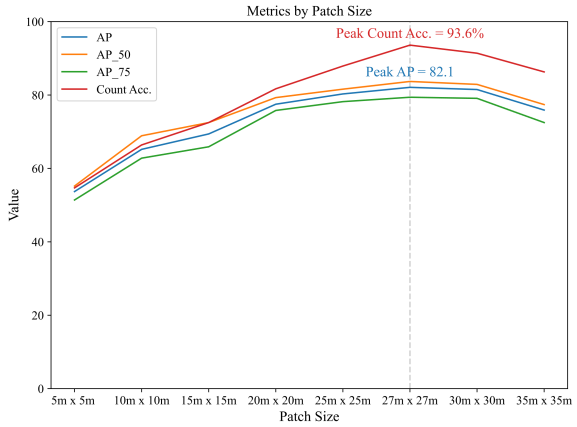

Figure A4: **Tuning for Image Patch size.** Patch size vs. AP and count accuracy.

latter we had to optimize for both equations. This resulted in a reduction of 42% in efficiency (i.e., lower tiles/minute throughput) while having the same instance segmentation performance (AP). Therefore, we choose to optimize for Eq. 8 in the main paper's approach.

A8 Dataset Preview

To the best of our knowledge, there exists no dataset of annotated informal settlements of underserved communities. To compound this, there exists only a population count of such areas instead of officially reported settlements. Therefore, not only is our work novel, but the dataset will contain annotations for 0.5mpp satellite imagery of informal settlements in five countries (Brazil, Mexico, India, Pakistan, and Kenya) spanning three continents (South America, Asia, and Africa). The approximate average density of settlements in these areas is an astounding 54,640 settlements/favelas/shanties per square kilometer. We have illustrated a collection of images alongside key statistics like precise locations, densities, and extents.

Figure A5: **Additional Qualitative Results.** Comparisons of our approach vs. DiffusionDet, Mask2Former, SAM 2, and MS-RCNN. Similar to the illustrative results in the main paper's baselines, it is observed to even a larger degree that our method performs instance segmentation closer to GT whereas the other baselines either miss a significant number of instances or merge multiple instances into one.

Methods	Brazil			Mexico			India			Pakistan			Kenya		
	GT	Pred.	Acc.↑	GT	Pred.	Acc.↑	GT	Pred.	Acc.↑	GT	Pred.	Acc.↑	GT	Pred.	Acc.↑
LOGCAN++	1924	1679	87.3	1692	1388	82.0	398	309	77.6	475	348	73.3	486	359	73.9
MaskRCNN (Swin-T)	1924	1524	79.2	1692	1324	78.3	398	302	75.9	475	335	70.5	486	342	70.4
SAM2	1924	1196	62.1	1692	1081	63.9	398	251	63.1	475	317	66.7	486	324	66.7
DiffusionDet	1924	1233	64.1	1692	1055	62.4	398	244	61.3	475	306	64.4	486	311	64.0
MS-RCNN (ResNet-101)	1924	1442	74.9	1692	1348	79.7	398	304	76.4	475	368	77.5	486	361	74.3
Mask2Former	1924	1247	64.8	1692	1107	65.4	398	264	66.3	475	315	66.3	486	325	66.9
BoundaryFormer	1924	1626	84.5	1692	1421	84.0	398	315	79.1	475	381	80.2	486	376	77.4
AFM2Mask	1924	1508	78.4	1692	1394	82.4	398	312	78.4	475	377	79.5	486	353	72.8
SegRefiner	1924	1591	82.7	1692	1406	83.1	398	298	74.9	475	388	81.7	486	379	78.1
U2Seg	1924	1546	80.4	1692	1426	84.3	398	315	79.2	475	399	84.2	486	398	81.9
MCAC	1924	1532	79.6	1692	1435	84.8	398	311	78.1	475	374	78.7	486	401	82.5
Ours	1924	1801	**93.6**	1692	1492	**88.2**	398	335	**84.2**	475	417	**87.8**	486	441	**90.7**

Table A1: **Counting Performance.** Comparisons of counting performance to prior state-of-the-art baselines. Here Acc. is defined as a percentage of normalized $1 - MAE$.

Method	AP↑	Acc.↑
Ours with 10% Perturbation	81.6	92.5
Ours with 20% Perturbation	80.2	89.7
Ours (w/ Faster RCNN)	73.9	81.6
Ours (w/ FCOS)	75.3	84.2
Ours (w/ Our Center Generator)	**82.1**	**93.6**

Table A2: **Center Extractor Comparisons.** Experiments with different center extractors showing higher AP, count accuracy (Acc.), and efficiency of our center extractor to SOTA tracking model center extractors.

Tested on→ Trained on↓	0.5 mpp (SkySat)	3 mpp (Dove)	30 mpp (Landsat)
0.5 mpp (SkySat)	82.1	75.2	50.3
3 mpp (Dove)	77.2	78.2	52.7
30 mpp (Landsat)	53.5	52.2	54.7
Combined	78.1	77.4	63.5

Table A3: **Resolution Varied Cross-validation (AP).** Segmentation performance of training/testing on standardized resolutions.

A9 Implementation Details

Computing Resources. All experiments and training were conducted on a system equipped with an AMD EPYC 7302 16-core processor operating at a clock speed of 3 GHz and 128 GB of RAM. Model training was performed using four NVIDIA RTX 3090 GPUs in parallel.

Our work uses the PyTorch framework. For graph processing tasks, we employ the PyTorch Geometric (PyG) library, while OpenCV is utilized for image processing. The deep point extraction component is built upon a ResNet-18 (He et al. 2016) backbone, with its output serving as input prompts to a SAM 2 network (Ravi et al. 2025) during the initial stage of processing. Furthermore, our graph-based model incorporates the GINEConv layer (Hu et al. 2020) from PyTorch Geometric to perform graph isomorphic convolution with edge features.

Patch Size	AP ↑	AP_{50} ↑	AP_{75} ↑	Count Acc. ↑
5m × 5m	53.7	55.2	51.4	54.7
10m × 10m	65.2	68.9	62.8	66.4
15m × 15m	69.4	72.5	65.9	72.5
20m × 20m	77.5	79.3	75.8	81.7
25m × 25m	80.3	81.6	78.2	87.9
27m × 27m	**82.1**	**83.7**	**79.4**	**93.6**
30m × 30m	81.5	82.9	79.1	91.4
35m × 35m	75.9	77.4	72.5	86.3

Table A4: **Patch Size Hyperparameter Tuning.** Calculation of segmentation performance and counting accuracy to find the optimal image patch size.

Training. For the first phase of point extraction, the network was trained for approximately 4 hours using a base learning rate of 0.001, a weight decay of 0.0005 over 200 epochs, and The Adaptive Moment Estimation (ADAM) optimizer. This training enabled the network to identify the initial center points of buildings, which, while prone to oversegmentation, served as input prompts for the SAM 2 network in phase 2 to execute image oversegmentation.

In phase 3, our GINEConv (Hu et al. 2020) network was used to perform message passing between node and edge embeddings. Input features were designed to encode each node's local neighborhood information. In this implementation, information aggregation was limited to one-hop neighbors, ensuring each node embedding captured only immediate relational data. The network was trained using a base learning rate of 0.001 and a weight decay of 0.0005, optimized with ADAM over 4,000 epochs, as convergence required a significant number of training iterations. Training on the dataset took approximately 7 hours. Early stopping was not used and instead the optimization was periodically inspected to ensure no overfitting.

Baselines. We document the training and usage of the listed baseline models on our dataset.

MaskRCNN with SWIN-T backbone (Liu et al. 2021b). We used the official implementation available on Microsoft Research's GitHub at https://github.com/microsoft/Swin-Transformer, which extends MaskRCNN with the Swin-T backbone for instance segmentation. The dataset was converted into COCO format for compatibility with the repository. The model was trained using ADAM with a base learning rate of 0.0001 and a weight decay of 0.0001. Training for 150 epochs took approximately 36 hours.

BoundaryFormer (Lazarow, Xu, and Tu 2022). The official implementation from UCSD's GitHub repository at https://github.com/mlpc-ucsd/BoundaryFormer was used to train BoundaryFormer on our dataset. The training process utilized ADAM with a base learning rate of 0.002 and weight decay of 0.0005. Training was conducted for 200 epochs, requiring approximately two days to complete.

SAM 2 (Ravi et al. 2025). We used the official implementation available at https://github.com/facebookresearch/sam2. SAM 2 is a generalized segmentation model designed to handle diverse datasets. For our work, we utilized the pre-trained weights provided by the authors, running the model in its "everything" mode to generate over-segmentation masks. This approach required no fine-tuning, as the generalized nature of SAM 2 allowed it to adapt effectively to our data without additional training. The inference process was efficient, taking approximately 2 hours to process our entire dataset on our quad NVIDIA RTX 3090 GPU.

Mask Scoring RCNN (Huang et al. 2019). The baseline was implemented using the official GitHub repository at https://github.com/zjhuang22/maskscoring_rcnn. The model was retrained on our dataset, formatted for COCO compatibility. SGD with a base learning rate of 0.02, momentum of 0.9, and weight decay of 0.0001 was used as the optimizer. The training lasted for 100 epochs, completing in roughly one day.

Mask2Former (Cheng et al. 2022). We employed the official implementation from Meta's GitHub repository at https://github.com/facebookresearch/Mask2Former. The model was trained on our dataset for 100 epochs with ADAM using a base learning rate of 0.0001 and a weight decay of 0.05. The training process took approximately eight hours.

DiffusionDet (Liu, Ren, and Zhao 2023). The DiffusionDet baseline was implemented using the official repository at https://github.com/ShoufaChen/DiffusionDet. We retrained the model on our dataset for 100 epochs using ADAM with a base learning rate of 0.0002 and weight decay of 0.0001. The training process required approximately 10 hours to complete.

LOGCAN++ (Ma et al. 2025). The implementation of LOGCAN++ was based on the official repository available at https://github.com/xwmaxwma/rssegmentation.The model was trained using our dataset in COCO format. Training was done using SGD with a base learning rate of 0.01, momentum of 0.9, and a weight decay of 0.0005. The training ran for 150 epochs and took around 18 hours.

AFM2Mask (Li et al. 2022). The AFM2Mask baseline was implemented from the authors' original repository given in https://github.com/lqycrystal/AFM_building/. We retrained the model using a dataset formatted for COCO compatibility. The training setup used the Unet backbone (as per the original paper) with a base learning rate of 0.02, momentum of 0.9, and a weight decay of 0.00001. We used the original configuration of 16 tiles per batch. The training ran with maximum epochs set to 90,000 with early stopping and it took approximately 12 hours.

SegRefiner (Wang et al. 2023). We used the official implementation from https://github.com/MengyuWang826/SegRefiner. We used SAM2 to be the base segmenter and SegRefiner to refine it to keep every aspect parallel to our approach. We trained the model with our dataset in COCO format. ADAM optimizer was used with a learning rate of 0.0001 and a weight decay of 0.001. Further, the training was run for 50 epochs with 1,000 diffusion steps and took approximately 14 hours.

U2Seg (Wang et al. 2023). We used the U2Seg implementation based on the official repository available at https://github.com/u2seg/U2Seg. The model utilizes Detectron2 as its foundational framework and employs a Base-RCNN-C4 backbone for feature extraction. The training was conducted using our dataset formatted in COCO format, with SGD as the optimizer, a base learning rate of 0.02, momentum of 0.9, and a weight decay of 0.0005. The training ran for 90,000 iterations, with learning rate decay steps at 60,000 and 80,000 iterations. The batch size was set to 16 images per batch. It took approximately 12 hours for training.

MCAC-ABC123 (Hobley and Prisacariu 2024). We note that MCAC-ABC123 (Hobley and Prisacariu 2024) is not a segmentation model but a heatmap based counting model that is class agnostic. It facilitated us to use the popularly used heatmap regression based crowd counting approach on our dense object data. We used the official repository available at https://github.com/ActiveVisionLab/ABC123. Retraining was redundant in this instance as the method is a counting model and class agnostic. The inference and testing took approximately 11 hours on the weights of the MCAC dataset.