This supplementary is organized as follows:

- Sec. A sheds light on the principal thinking of city structures and provides the additional background on the pipeline of city modeling, from which we narrow down to the scope of city block modeling.
- Sec. B talks about the details related to *NYC-Block Dataset* collection, pre-processing, and extensions.
- Sec. C includes subsections relevant to the main experiments, including the implementation details of comparing methods, model architecture, metric designs, loss compositions, etc. More illustrated samples are attached at the end of the supplementary.
- Details with regard to various applications are explained in Sec. D; in Sec. E, we briefly discuss the limitations and future works.

A. Background

City Structures. To help readers better understand the linkages between city structure and block patterns, we first provide a series of overhead views of cities across the world. As shown in Fig. 13, the urban fabrics of different cities are formed by distinctive block patterns. Fig. 14 zooms in to focus on one specific city block, where a constructed city block is abstracted into a 2D block plan in canonical view, colored by land use categories with illustrated footprint shapes residing in each land lot.

City Modeling. The major steps of a typical city modeling task include: (i) Create the terrain and partition the city or urban area into functional zones; (ii) Generate an interconnected road network; (iii) Subdivide the space in-between roads, *i.e.*, city blocks, into small land lots of different land uses; and (iv) Populate tax lots with buildings, parks, and other urban structures based on their semantics. These series of steps shed insight on the structural nature of city area, where the city block is the fundamental element of the physical structure of urban areas; and city *block patterns* give order and structure to a city, forming the basic unit of a city's urban fabric, as Allan Jacobs describes:

> *"…city street and **block patterns** can **give order and structure** to a city, district, or neighborhood…All of this can happen at a **two-dimensional** level, without regard to a **third dimension** of topography or building height or to a **fourth dimension** that includes land uses and density of habitation - factors that can in themselves give order and structure, either reinforcing the two-dimensional patterns or running counter to them."*
> *– Great Streets* [1]

Serving as a foundation for city modeling, we focus on city block construction, *i.e.*, the (iii) and (iv) steps in the aforementioned pipeline. Our *BlockPlanner* system is designed to automate this step through a data-driven end-to-end learning paradigm.

[1]This excerpt from *Great Streets* [18], written by the renowned urban designer Allan Jacobs in 1993, reveals the multi-dimensional nature of urban design and city structures, where the concepts of *block pattern*, *land use*, *city order*, and *structure* are connected and emphasized.

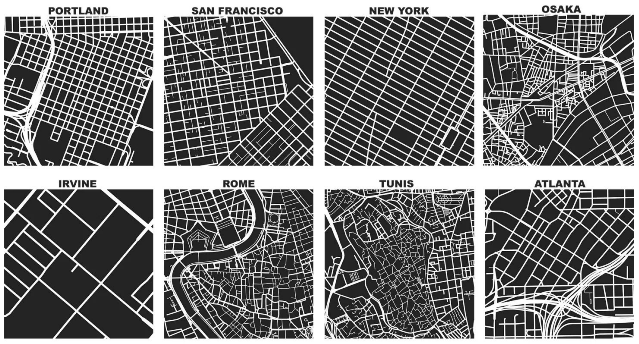

Figure 13: City block patterns visualized with OSMnx [7]. The square mile drawings reveal how city block patterns are wonderfully varied across cities, and that they are individually characteristic, highly regularized, and memorable [18].

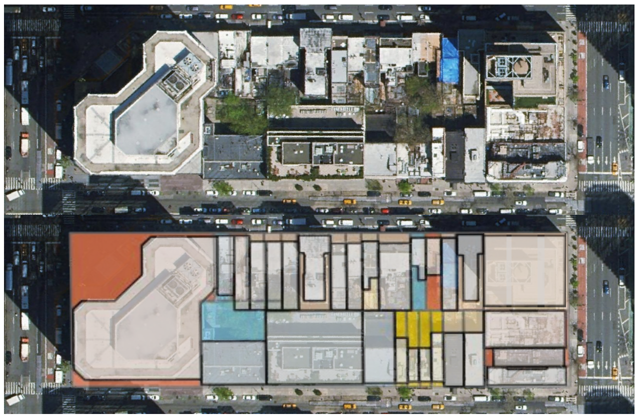

Figure 14: Satellite view of a city block, showing how a natural-looking city block image can be abstracted into a 2D city block plan. The block is divided into different land lots associated with land use attributes, which is the study target of this work.

B. NYC-Block Dataset

B.1. Data Preparation

Data Sources. PLUTO[2] provides extensive *land use* and *geographic data* at the tax lot level in ESRI shapefiles and File Geodatabase formats, covering the whole NYC area. The block shapes we used in our *NYC-Block dataset* are based on these shapefiles, followed by a series of canonicalization steps.

[2]PLUTO is a backronym that stands for Primary Land Use Tax Output, officially provided by NYC Department of City Planning. Notably, PLUTO is created using the best available data from a number of city agencies, fully open to the public, and updated timely with ensured data accuracy and reliability.

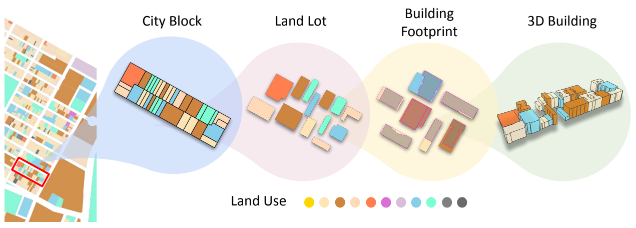

Figure 15: Terminologies related to city block.

Terminology. To better understand the task of city modeling, we elaborate on the key terms used in our work as illustrated in Fig. 15, as defined in urban planning and real estate: 1) The *city block* is the fundamental element of physical structure of urban areas. A *city block* is primarily a tract of land bounded on all sides by streets or by a combination of streets. It is defined by an edge and an interior; 2) The spatial configuration and layout of a city block are defined on the basis of its internal arrangement of *land lots*. Often a lot is sized for a single house or other building; 3) Each *land lot* is associated with a *land use* type. *Land use* is based on the function of the land for different humane purposes and economic activities. Typical *land use* types include residential, commercial, industrial, and transportation; 4) A *building envelope* is the maximum three-dimensional space on a *land lot*, within which a structure can be built following a city's regulation; 5) The *footprint of a building* is its perimeter at the outer edge of the outside walls of the building.

Map Projection. In PLUTO, land lots geometries are stored as polygons defined by the longitude and latitude of their vertices under World Geodetic System 1984 (WGS84), EPSG:4326. To visualize the WGS84 coordinate system on a two-dimensional plane, we first project the coordinates onto a square with the projected Pseudo-Mercator coordinate system, EPSG:3857, where the mapping is *conformal*[3].

Canonical View Transformation. To canonicalize the arbitrary rotations of city blocks, we calculate the angle between the long side of each city block and the horizontal, and perform rotation to the shape coordinates. We centralize the blocks and normalize them into a unit ball in 2D, where the height n is normalized to $[0, 1]$.

Selected Features. The PLUTO files contain more than seventy fields. In this work, we use the two most important features, *i.e.*, *land use* and *number of floors* representing building height limitation.

Land Use Statistics. Fig. 16 shows land use distribution over five NYC boroughs and the entire NYC respectively. It is noticeable that land use distribution in Manhattan appears to be the most diverse and complex amongst all five boroughs. Therefore, we choose to study the Manhattan subset in our main experiments.

[3]Mercator projection is unique in representing north as up and south as down everywhere while preserving local directions and shapes.

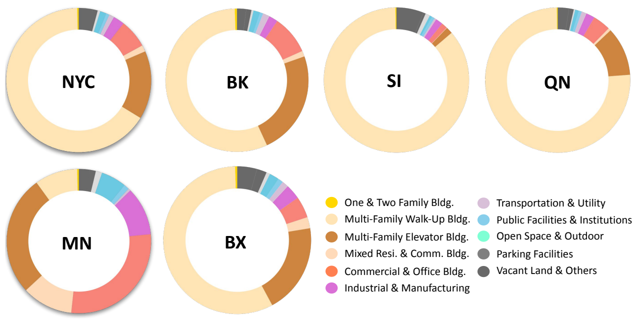

Figure 16: Land use distribution in the entire New York City (NYC) and five boroughs, Brooklyn (BK), Staten Island (SI), Queens (QN), Manhattan (MN), and Bronx (BX).

B.2. Dataset Scalability

A natural extension of our dataset is to include city blocks from a larger urban area. Additional to this, one extra merit of *NYC-Block Dataset* is that it can be easily fused with other datasets through attribute joins. We briefly introduce two ways to enrich our datasets.

Join by BBL. Borough, Block, and Lot (BBL) is the parcel number system used to identify each unit of real estate in New York City for numerous city purposes. For example, NYC DOITT [3] provides building information across the entire NYC, where each instance is associated with its land lot through BBL. Our *NYC-Block Dataset* can hence be enriched with building-related information by joining these two data sources via BBL. The enriched dataset can thus support further city modeling tasks, such as the footprint generation step we mentioned in the application section. The attributes recorded by NYC DOITT include: building footprints, ground elevation at building base, roof height above ground elevation, construction year, and feature type. A number of official city data related to zoning, finance, environments, politics, etc., can be joined in similar ways.

Join by Coordinates. A more general scenario is to establish linkage through the ubiquitous coordinates system, *i.e.*, latitude and longitude provided by PLUTO database. Besides the abundant *street view images* and *satellite images*, more and more social media texts, photos, and videos nowadays are tagged with geo-locations. The potential to link these visual images, human activity data, to the city regularization data provided by the government can be a valuable and exciting direction for future study.

Extension to More Cities. *NYC-Block Dataset* can be easily augmented to include other cities. Similar land use documents are made publicly available by many cities other than NYC, such as Washington D.C., Los Angeles, and London. Data from new cities can be integrated into the existing dataset following our systematic pipeline.

Algorithm 1 Graph Encoder.

```
Input: Block capacity $N$ (maximum number of lots in a block); Number of message passing iterations $T$; Lot geometry $(x_{i,c}, y_{i,c}, w_i, h_i, n_i)$, and land use category $s_{i,l}$ for $L_i, i \in \{0, 1, \ldots, N-1\}$ ; Edge matrix $\mathcal{E}$.
Output: Block code $z$
  Encode lot geometry and semantics
  $f_{i,g} = e_g(\{x_c, y_c, w, h, n\})$, $f_{i,s} = e_s(\{s_l\})$
  Ensemble $f_{i,g}, f_{i,s}$ with one-hot position encoding $f_{i,PE}$
  $f_i^{(0)} = f_i = e_{lot}(f_{i,g}, f_{i,s}, f_{i,PE})$
  $z^{(0)} = \text{maxpool}(\{f_i^{(0)}\})$
  for $t = 0$ to $T-1$ do
    for $e_{i,j}$ in $\mathcal{E}$ do
      $f_{i,j}^{(t)} = h^{(t)}(f_i^{(t)}, f_j^{(t)})$
    end for
    Updated lot feature $f_i^{(t+1)} = \frac{1}{N} \sum_{e_{i,j} \in \mathcal{E}} f_{i,j}^{(t)}$;
    Obtain block feature $z^{(t+1)} = \text{maxpool}(\{f_i^{(t+1)}\})$;
  end for
  Aggregate block feature $z = g_a\{z^{(0)}, \ldots, z^{(T)}\}$;
  return $z$
```

C. Additional Implementation Details

C.1. Implementation of Comparing Methods

To make the comparing methods compatible with city block generation scenario, the following adjustments are made:
LayoutVAE. [20] We re-implement their method and experiment on our city block dataset.
MolGAN. [11] We modify it into a VAE-based framework, termed MolVAE, while preserving their original network architectures. We find that the training of MolVAE is more stable on city blocks compared to its GAN counterparts.
HouseGAN. [27] Based on HouseGAN model, we make the following modifications and term it as BlockGAN in our experiments: 1) We eliminate its dependency on input graph, transform it into an unconditional generative framework. 2) We treat each land use assembly as an instance, output a binary mask for each land use; the visual quality is then evaluated without rendering the lot instance edges. 3) To enforce the boundary constraints (*i.e.*, ensure all land lots are resided within the block), we add an additional channel for a binary block boundary mask, stacked behind all land use channels and pass to the discriminator to ensure that all the generated land lots reside within the 2D block envelope. 4) The rasterized image resolution is set to 256×256 in order to capture all land lots shapes.

C.2. Network Details

Following the recent success of generative models on graphs [11, 25], we train our *BlockPlanner* under the variational scheme. The adopted training schemes and network designs for the graph encoder/decoder part adopted are explained in Alg. 1 (Fig. 17) and Alg. 2 (Fig. 18) respectively. The detailed architecture parameters are specified in Tab. 4.

Algorithm 2 Graph Decoder.

```
Input: Block code $z$; Block capacity $N$; Number of message passing iterations $T$.
Output: Lot geometry $(\tilde{x}_{i,c}, \tilde{y}_{i,c}, \tilde{w}_i, \tilde{h}_i, \tilde{n}_i)$, land use category $\tilde{s}_{i,l}$, boundary attribute $\tilde{s}_{i,b}$, merge operation $\tilde{s}_{i,m}$ for $L_i, i \in \{0, 1, \ldots, N-1\}$ ; Edge matrix $\tilde{\mathcal{E}}$; Block aspect ratio $\tilde{r}_y$.
  Decode block aspect ratio $\tilde{r}_y = d_r(z)$
  Decode edge probability map $\tilde{\mathcal{P}} = d_{edge}(z)$
  Obtain the edge matrix $\tilde{\mathcal{E}} = \tilde{\mathcal{P}} \geq 0.5$
  Initialize lot feature $\tilde{f}_i^{(0)} = \{d_{lot}(z), f_{i,PE}\}$
  for $t = 0$ to $T-1$ do
    for $\tilde{e}_{i,j}$ in $\tilde{\mathcal{E}}$ do
      $\tilde{f}_{i,j}^{(t)} = h_d^{(t)}(\tilde{f}_i^{(t)}, \tilde{f}_j^{(t)})$
    end for
    Updated lot feature $\tilde{f}_i^{(t+1)} = \frac{1}{N} \sum_{\tilde{e}_{i,j} \in \tilde{\mathcal{E}}} \tilde{f}_{i,j}^{(t)}$
  end for
  Assign $\tilde{f}_i = \tilde{f}_i^{(T)}$;
  Decode lot exists probability $p_i = \sigma(g_{lot}(\tilde{f}_i))$;
  Decode lot geometry $(\tilde{x}_{i,c}, \tilde{y}_{i,c}, \tilde{w}_i, \tilde{h}_i, \tilde{n}_i) = g_{box}(\tilde{f}_i)$;
  Decode land use attribute $\tilde{s}_l = g_{label}(\tilde{f}_i)$;
  Decode lot boundary attribute $\tilde{s}_b = g_{bound}(\tilde{f}_i)$;
  Decode lot merge operation $\tilde{s}_m = g_{merge}(\tilde{f}_i)$;
  return $p_i, (\tilde{x}_{i,c}, \tilde{y}_{i,c}, \tilde{w}_i, \tilde{h}_i, \tilde{n}_i), \tilde{s}_l, \tilde{s}_b, \tilde{s}_m, \tilde{\mathcal{E}}, \tilde{r}_y$
```

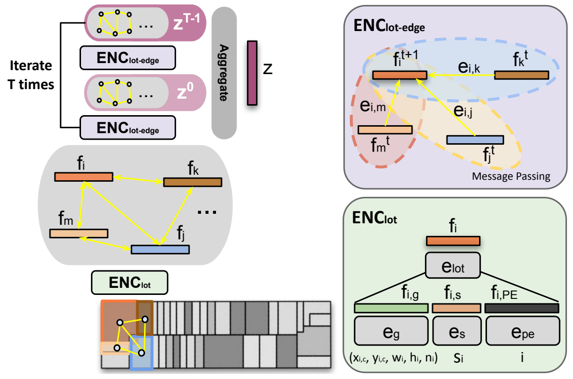

Figure 17: Illustration of the graph encoder.

C.3. Experimental Settings

We use PyTorch for implementation and a workstation with a single GTX TITAN X GPU. The base model sets the iteration number $T = 3$ for both encoder and decoder, with latent code dimension fixed as 256. We use Adam optimizer with $lr = 1e^{-3}$ for the first 200 epochs and schedule $lr = 1e^{-4}$ for another 200 epochs. The batch size is set to 32.

Architecture	Module	Layer
Encoder	lot_geometry_encoder (e_g)	Linear(5, 256)
	lot_semantic_embedding (e_s)	Embedding(11, 256)
	lot_encoder (e_{lot})	Linear(256+256+N, 256)
	node_edge_fuse_encoder ($h^{(t)}$)	Linear(256+256, 256)
	block_aggregate_encoder (g_a)	Linear(256*(T+1), 256)
Decoder	block_box_decoder (d_r)	Linear(256, 256); Linear(256, 1)
	edge_decoder (d_{edge})	Linear((256+N)*2, 256) ; Linear(256, 1)
	lot_initial_decoder (d_{lot})	Linear(256, 256*N)
	node_edge_fuse_decoder ($h_d^{(t)}$), $t = 0$	Linear((256+N)*2, 256)
	node_edge_fuse_decoder ($h_d^{(t)}$), $t > 0$	Linear(256+256, 256)
	node_exists_decoder (g_{lot})	Linear(256*N,1)
	landuse_head_layer (g_{label})	Linear(256, 11)
	bound_head_layer (g_{bound})	Linear(256, 4)
	merge_head_layer (g_{merge})	Linear(256, 2)

Table 4: Network Architectures. The default latent dimension is set to 256; N represents the block capacity (maximum number of lots in a block) and T represents the number of iterations used in GNN message passing.

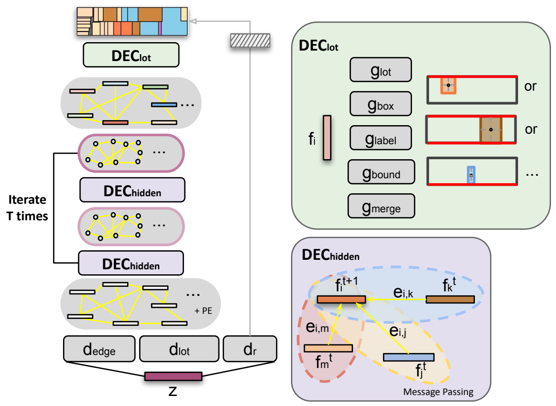

Figure 18: Illustration of the graph decoder.

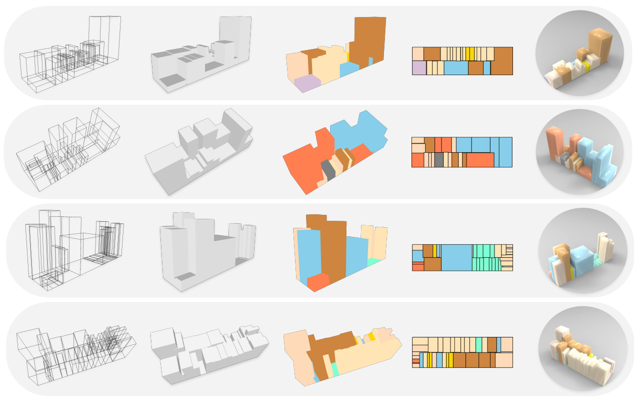

Figure 19: More generated samples with *BlockPlanner*, visualized in 3d software.

Geometry Validation. 1) *Boundary alignment* is validated by comparing the true boundary and the decoded boundary. Suppose $\tilde{s}_b = 0$ with decoded geometry $(\tilde{x}_{i,c}, \tilde{y}_{i,c}, \tilde{w}_i, \tilde{h}_i)$, then the violation is calculated by $| - 1 - (\tilde{x}_{i,c} - \tilde{w}_i/2)|$, where -1 is the true aligned boundary for x axis for $s_b = 0$. 2) *Adjacency validation* is compared between the paired lot geometries. For $\tilde{e}_{i,j} = 1$, the violation is calculated via $\max(|\tilde{x}_{i,c} - \tilde{x}_{j,c}| - (w_i/2 + w_j/2), 0)$, and vise versa.

Loss Weights. The weight of each loss term is empirically set to $w_r, w_x, w_s, w_g, w_v = 10 : 5 : 2 : 1 : 2$. We find this setting achieves good results with relative faster convergence. Note that, the impact of weights is not remarkable when $\mathcal{L}_{adj}$ and $\mathcal{L}_{bound}$ are imposed.

C.4. Remarks on Evaluation Metrics

Realism. Realism is measured by an average user rating on the visual fidelity. We carry out a user study with 10 amateurs and 10 professionals with rasterized 2D plan images. Samples generated by different methods are mixed with the real building blocks (100 samples each) for users to distinguish. For each sample, the user is asked to give a score in $[0, 1, 2, 3, 4, 5]$, where 5 stands for a good and realistic looking, and 0 for unrealistic looking and unreasonable configuration. The score for each sample is further normalized to $[0, 1]$, where the final score takes the summation over each sample, expanding from 0 to 100 for each user. Our report Realism is the average final score over all users.

FID. We adopt FID score [14] to measure the visual quality and diversity on 2D rasterized block layout, which is widely used in floorplan generation evaluations [27, 28]. 600 random fake samples are generated for each method, and rasterized in 2D. For unfilled areas, we treat them as vacant land use types and filled with darkgray color. Except for BlockGAN, all land lots boxes are colored with black boundaries to highlight lot instances, and compare with the corresponding ground truth images with lot boundaries. For

BlockGAN, we render the image purely with colors to differentiate land use types, and compare with corresponding ground truth images without lot boundaries.

Land Use Statistics. The generated blocks should follow the overall land use distribution of the real distribution. Apart from the basic count distribution, *i.e.*, the number of lots per block, and the total portion of lots over land use categories, we also measure the *one-step transition probability* that approximates a human's experience of what one will arrive from his current place to the neighboring lot [4].

D. Applications

Topology Refinements. The set of refinement parameters X that we mentioned to control the translation (dx, dy) and scaling magnitude (sx, sy) of each lot are as follows,

$$X = (X_{dx}, X_{dy}, X_{sx}, X_{sy}) \in \mathbb{R}^{N\times 4}. \tag{8}$$

For lot with original generated shape (x_c, y_c, w, h, n), its final 2D geometry is determined by,

$$(x_c + dx, y_c + dy, w(1+sx), h(1+sy)). \tag{9}$$

The optimal parameter can be obtained by minimizing:

$$\min_X D_{gap}(r(\mathcal{G})) + \lambda |X|_2. \tag{10}$$

where $r(\cdot)$ indicates the rendered layout, and $D_{gap}(\cdot)$ is the accumulative gap areas. It also calculates the violation to predicted order in the ring topology, including the too much portion in the overlapping area. We use $\lambda|X|_2$ to constrain the magnitude of changes, reflecting our desire to ask each lot to maintain its original geometry as much as possible. In practice, we use SGD optimizer in PyTorch and early-stop the iteration once the objective loss is below a preset tolerance ϵ. In fact, users are free to choose proper objective functions, variables, or constraints to realize different editing goals by means of optimization.

Footprint Generation. In the main paper, we showcase one-step further to synthesize plausible building footprints upon the generated block plans, *i.e.*, to generate 3D building models that reside in each land lot. However, simply replacing each generated lot with an independently selected footprint from a fixed collection is neither ideal nor practical. a strong correlation among footprint shapes within neighboring lots in a block can be observed given the land use type. We wish to generate a plausible footprint for each lot within block content conditioned on land use category. The task is

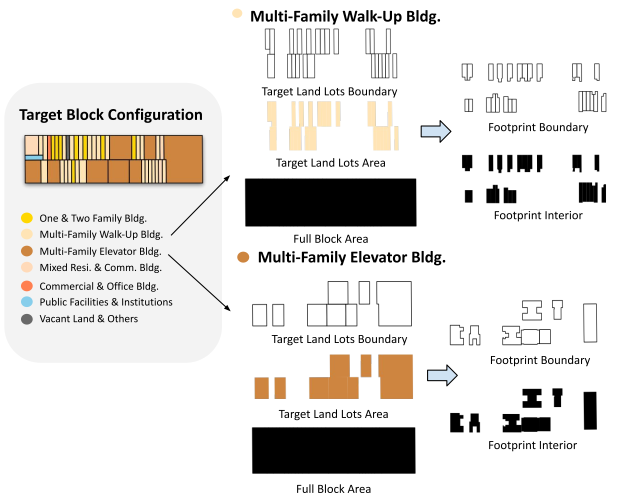
Multi-Family Walk-Up Bldg.
Target Block Configuration
● One & Two Family Bldg.
● Multi-Family Walk-Up Bldg.
● Multi-Family Elevator Bldg.
● Mixed Resi. & Comm. Bldg.
● Commercial & Office Bldg.
● Public Facilities & Institutions
● Vacant Land & Others
Target Land Lots Boundary
Target Land Lots Area
Full Block Area
Footprint Boundary
Footprint Interior
Multi-Family Elevator Bldg.
Target Land Lots Boundary
Target Land Lots Area
Full Block Area
Footprint Boundary
Footprint Interior

Figure 20: Illustration of footprint generation pipeline.

formulated as a classical image-to-image translation problem conditioned on the generated city blocks. An illustration is given in Fig. 20. We construct over 600 paired block data for training, and adopt pix2pix [17] for demonstration. The input image is the concatenation of lot bounding boxes and the land use semantic map; and the output is the refined footprint binary mask for each land use assembly.

From Manhattan to Brooklyn. Deep Network Interpolation (DNI) [34] is a simple yet universal approach for smooth and continuous imagery effect transition. We let N_{MN} with parameters θ_A be the network trained on Manhattan blocks, and finetune N_{MN} on Brooklyn blocks for another 100 epochs and denote as N_{BK} with parameters θ_A. The smooth transition is achieved by interpolating the model parameters across all layers with

$$\theta_{\text{interp}} = \alpha\theta_A + (1-\alpha)\theta_B. \tag{11}$$

Extension to Indoor Scenes. Indoor scene is a natural extension to block plan generation, as they share many design principles in common. While the land use semantics can be directly replaced with room types, we made the following adaptation to the ring topology: the peripheral rooms are denoted with $s_b \in \{0, 1, 2, 3\}$ as usual; the ones that reside in house interior are marked with $s_b = 4$. Corresponding geometry validations are then added for those interior rooms.

E. Discussion and Future Works.

BlockPlanner is a brand new trial on city block modeling with comprehensive consideration on block configuration and land uses, learned with generative models. A lot of interesting applications and extensions could be derived from.

Extension to larger scale. As the study unit of *BlockPlanner* is individual blocks, the first-tier node in our current graph representation only contains one element, serving as

[4] *Accessibility* plays an important role in urban planning. Besides the "street accessibility" we mentioned to introduce the lot attribute s_m, here we consider another type of accessibility in neighborhood evaluation.

If you are living in a block where its main land uses are residual houses, your neighboring lots are likely to be residential building or some mix residual and commercial land lots, such as the local grocery, to support your daily consumer demands; while it is unlikely or even impossible to have industrial land lots as your neighbors.

the summary node for a block. This treatment can be naturally extended to larger contents. For example, we can allow multiple first-tier nodes to capture the city layouts at the *district* or *borough* level, and even extend to multi-tier graph representation for larger area urban modeling.

Generalization to new block styles. The hypotheses on ring topology are set to model blocks in NYC. However, it is a flexible setting where constraints can be adjusted to accommodate new block styles. 1) Firstly, most city blocks can be abstracted into a ring topology. By definition, a city block is a group of buildings surrounded by streets with most of the buildings directly accessible from streets. Thus the ring topology well reflects this property and thus is a suitable assumption for city block modeling. 2) As also shown in Sec. 6.2 in the main text, our method also generalizes well to irregular structure such as indoor floorplans, which reveal that the ring topology assumption is not a hard constraint, but a flexible one that allow certain relaxation. For example, open spaces inside blocks can be treated as a special lot category labeled as *empty space*; and the number of boundaries n can be modified along with geometric constraints to model city blocks with polygonal shapes.

Extension to further city modeling steps. As mentioned earlier, our city block data can be conveniently joined with other data, such as the resourceful street view images tagged with geo-locations. It is therefore possible to extract building image textures directly from street view images, and well aligned with the block models. With such paired data, more interesting learning tasks can therefore be formulated. This may complete the auto-texturing step in city modeling, realizing vivid city block models for visualization, and provide the foundation towards more realistic renderings.

Adaptation to other cities. The proposed *BlockPlanner* system can be directly applied to cities other than NYC. While not all the cities have public open data that includes enough information as that in PLUTO, various alternative sources can be utilized: Google Maps, OpenStreetMap, etc.

Applications beyond Virtual Environments. *BlockPlanner* can serve as a powerful tool for designers to improve/redesign the current block 1) In real scenarios, urban designers always need to manually generate large amounts of proposals before finalizing. To avoid such challenging and tedious work, *BlockPlanner* provides a novel tool to obtain large amounts of diverse and valid block proposals automatically, and the generated blocks in vector format can get further edited and visualized in other tools. 2) We can also improve an existing block by shifting its latent code towards a desired direction, such as re-zoning from commercial to residential use, or interpolating between different block styles.

Limitations and future works. While our experiments demonstrate good performance on synthesizing city blocks under the canonical view, it will be our future work to make *BlockPlanner* more versatile in handling a broader range of city blocks in the real world. It is also interesting to experiment with more incorporated block features, especially for those social and environmental indexes to reveal their underlying connection, thus helping us better understand our living city through the machine eyes.

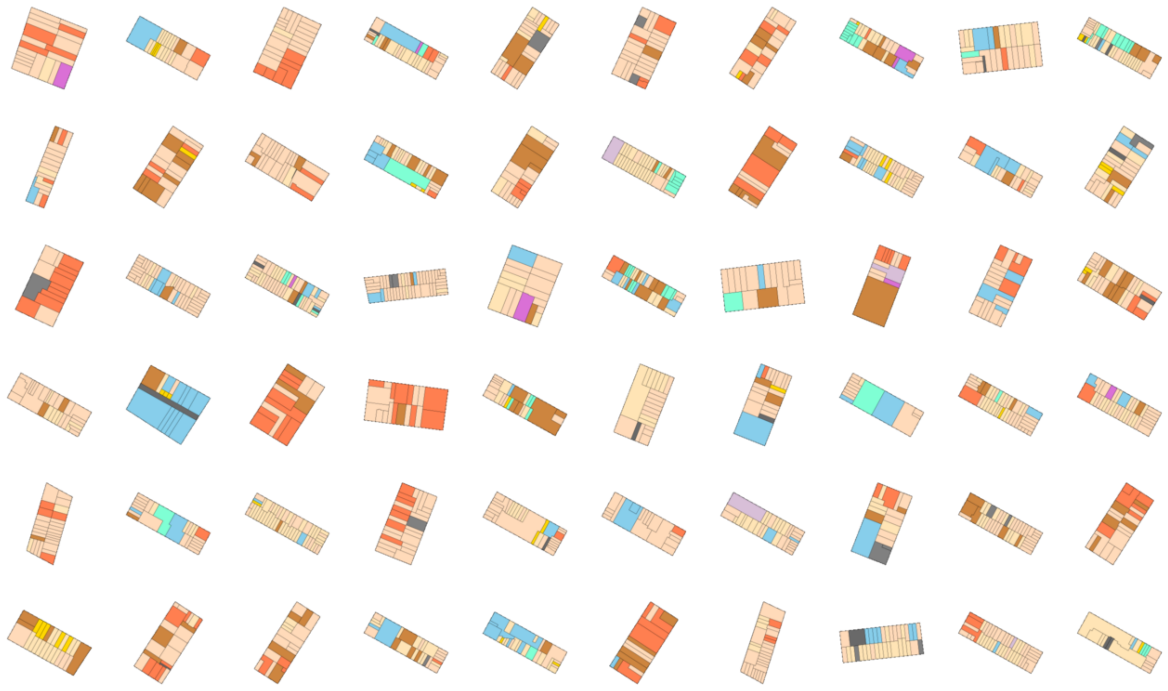
Figure 21: Sample Manhattan blocks extracted from PLUTO without canonicalization.

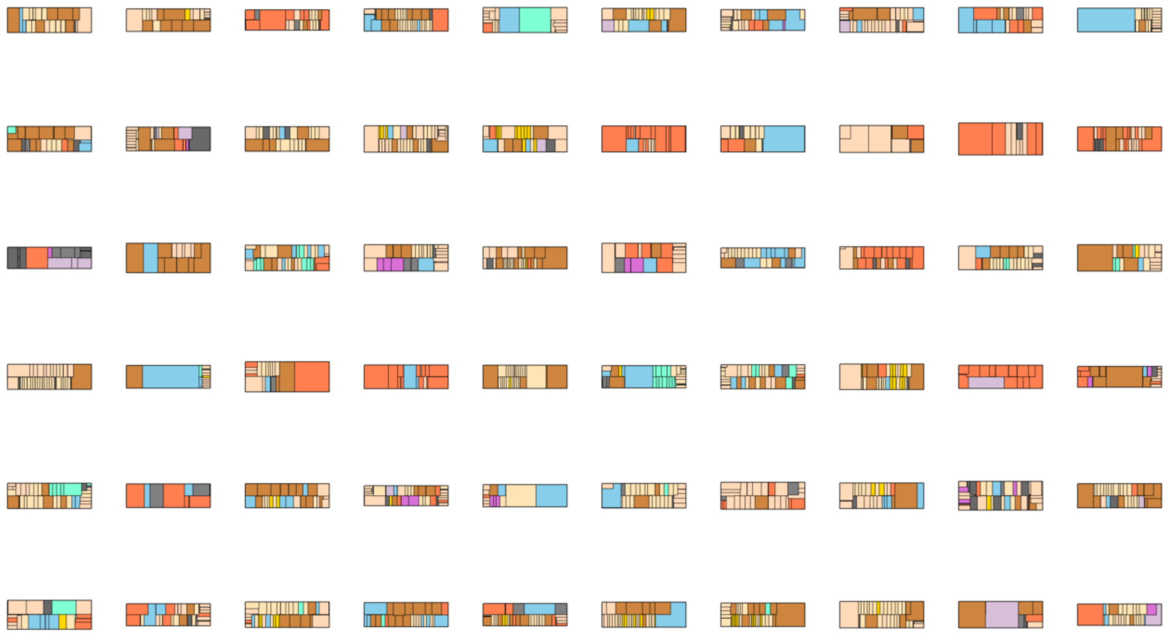
Figure 22: More generated results on BlockPlanner.

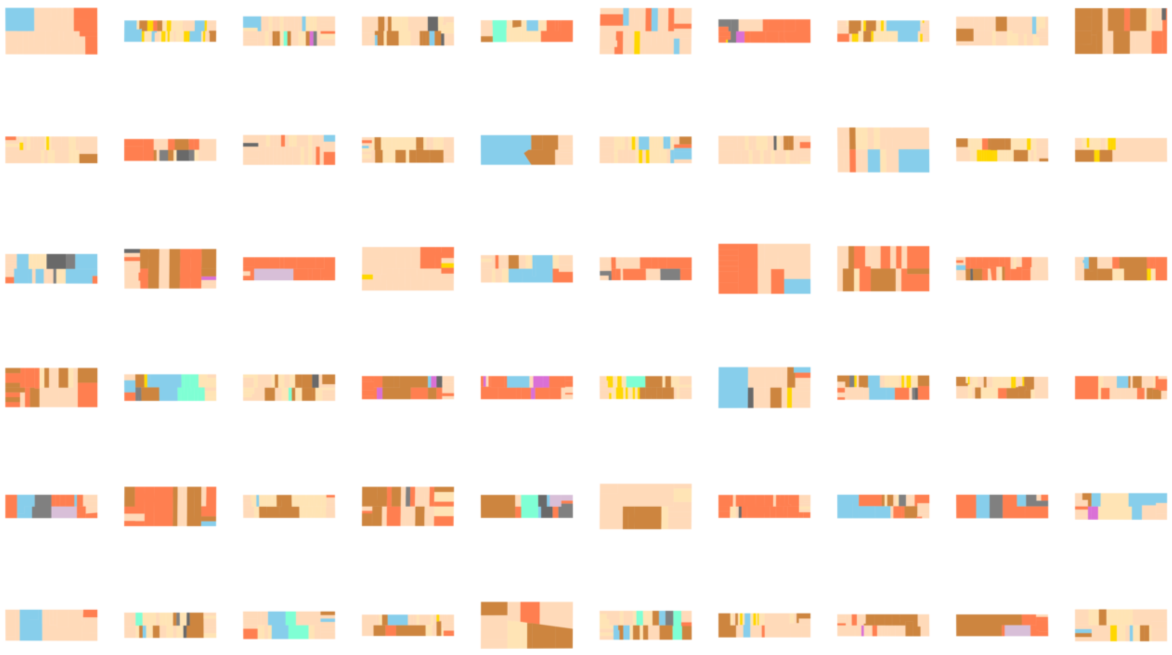
Figure 23: Real samples without the annotated lot boundary lines.

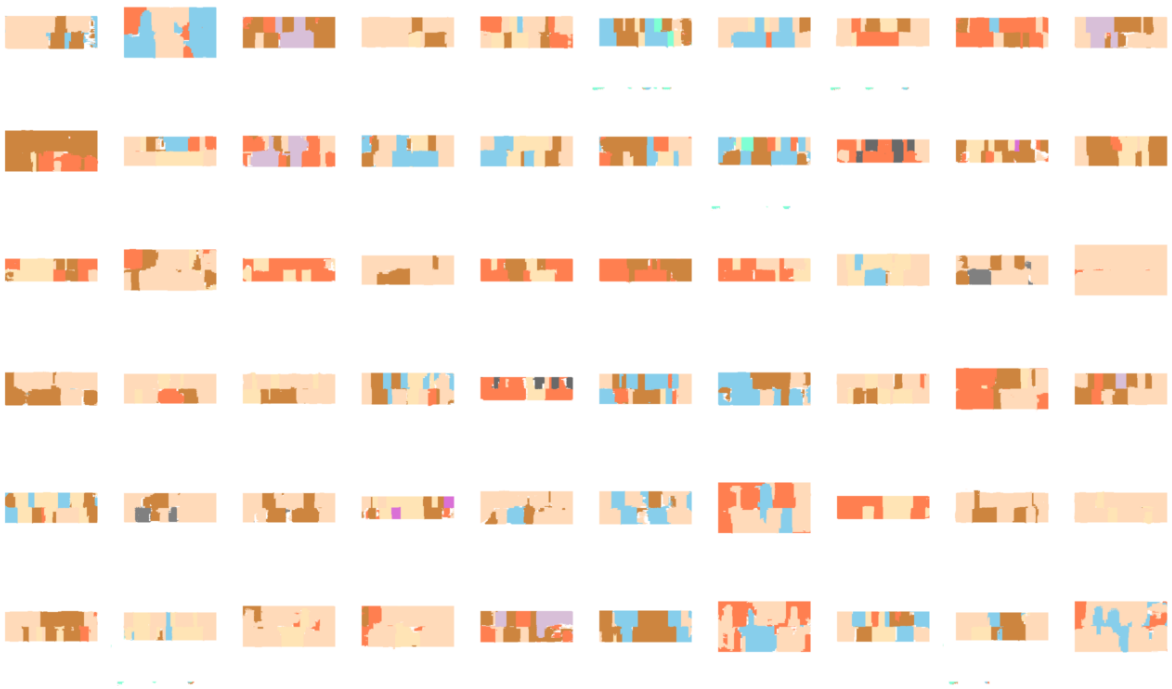
Figure 24: Generated samples with BlockGAN (with fuzzy boundary).