Fine-Grained Urban Flow Inference

Kun Ouyang, Yuxuan Liang, Ye Liu, Zekun Tong, Sijie Ruan, Yu Zheng, *Senior Member, IEEE* and David S. Rosenblum, *Fellow, IEEE*

Abstract—Spatially fine-grained urban flow data is critical for smart city efforts. Though fine-grained information is desirable for applications, it demands much more resources for the underlying storage system compared to coarse-grained data. To bridge the gap between storage efficiency and data utility, in this paper, we aim to infer fine-grained flows throughout a city from their coarse-grained counterparts. This task exhibits two challenges: the spatial correlations between coarse- and fine-grained urban flows, and the complexities of external impacts. To tackle these issues, we develop a model entitled UrbanFM which consists of two major parts: 1) an inference network to generate fine-grained flow distributions from coarse-grained inputs that uses a feature extraction module and a novel distributional upsampling module; 2) a general fusion subnet to further boost the performance by considering the influence of different external factors. This structure provides outstanding effectiveness and efficiency for small scale upsampling. However, the single-pass upsampling used by UrbanFM is insufficient at higher upscaling rates. Therefore, we further present UrbanPy, a cascading model for progressive inference of fine-grained urban flows by decomposing the original tasks into multiple subtasks. Compared to UrbanFM, such an enhanced structure demonstrates favorable performance for larger-scale inference tasks.

1 Introduction

Modern ubiquitous technology has provided big data of urban flows (e.g., traffic flow and population flow), which has been playing critical roles in smart city efforts. For example, government exploits human flow data to guide the planning of urban security resources such as police stations and CCTVs. Such data, especially the fine-grained ones, is indispensable for accurate decision making in city development.

To represent urban flows, a spectral of large-scale industrial systems [2, 3] uses grid map as the basic data structure, where each entry denotes the flow volume of agents in the corresponding area partition (see Figure 1 for example). The more partitions one rasterizes the urban area, the higher spatial resolution one can maintain in the data. While finer granularity is usually desirable for applications, it also results in quadratic growth of storage space (e.g., 128x128 resolution is 64 times larger than 16x16) and thus severely increases the resources demanded by the storage system (at TB level in typical industrial systems [4]). To trade off for system efficiency, one direct mitigation is to aggregate the fine-grained flow data and keep only its coarse-grained counterpart. Nevertheless, value of the data also degrades along with the loss of information granularity [5].

To bridge the gap between efficiency of storage systems and data usability for applications, in this paper, we present a technique to infer fine-grained urban flow data from its corresponding coarse-grained counterpart, so that we can store data in low granularity and recover fine-grained urban flow data whenever needed. Such Fine-grained Urban Flow Inference (FUFI) problem can be cast as a mapping problem that maps data of low information entropy to that of high information entropy, sharing the same nature as image super-resolution from the domain of image recovery. However, the FUFI problem exhibits two very specific challenges:

- **Spatial Correlations.** Fine-grained flow maps have spatial and structural correlations with their coarse-grained counterparts. Essentially, the flow volume in a coarse-grained superregion (e.g., the campus), is distributed among constituent subregions (e.g., libraries, sports center) at the fine-grained level. This implies a crucial structural constraint (i.e., **spatial hierarchy** [6]): the sum of the flow volumes among subregions strictly equals that of the corresponding superregion, as shown in Figure 1. Furthermore, the flow in one region can be affected by the flows in the nearby regions, which will impact the inference for the fine-grained flow distributions over subregions. Methods failing to capture these considerations would exhibit poor performance.
- **External Factors.** The distribution of the flows in a given region is affected by various external factors, such as local weather, time of day, and special events. To understand such impacts, we present a real-world study in an area of Beijing as shown in Figure 2(a). On weekdays, (b) shows more flows occurring at 10 a.m. in the office area and attractions as compared to at 8 p.m. when residences

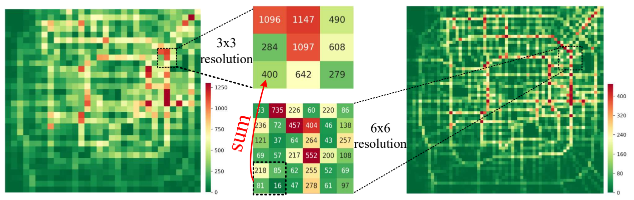

3x3 resolution
1096
1147
490
284
1097
608
400
642
279
sum
6x6 resolution
(a) Coarse-grained crowd flows (32x32)
(b) Fine-grained crowd flows (64x64)

Fig. 1. Real traffic flows at two levels of granularities in Beijing, where each grid denotes a region.

- *This paper is an extended version of an earlier paper published at the 25th SIGKDD conference (KDD 2019) [1]*
- *Kun Ouyang, Yuxuan Liang, Ye Liu, and David S. Rosenblum are with the School of Computing, National University of Singapore, Singapore 119077. E-mail:{ouyangk,yuxliang,liuye,david}@comp.nus.edu.sg*
- *Zekun Tong is with the Department of Industrial System Engineering, National University of Singapore, Singapore 119077. E-mail: vantong96@outlook.com*
- *Sijie Ruan and Yu Zheng are with JD Intelligent Cities Research & JD Intelligent Cities Research, Beijing, China. E-mail:{sijieruan,msyuzheng}@outlook.com*

Manuscript revised at June 24, 2020.

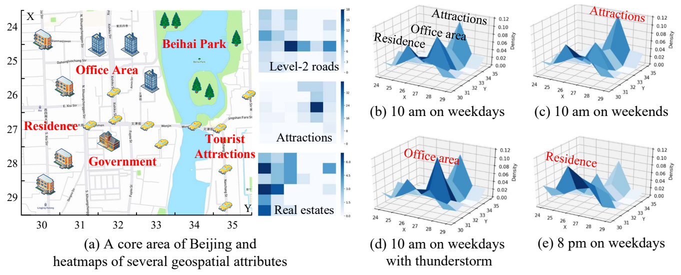


Fig. 2. Impacts of external factors on regional flow distributions. (a) We obtain Point of Interests (POIs) for different regions, and then categorize regions with different semantics according to the POI information. (b)-(e) depict the average flow distribution under various external conditions.

experience much higher flow density than the other areas (see (e)); on weekends, however, (c) depicts that people tend to be present in a park in the morning. All of these reflect our common sense that people go to work in the morning, to attractions for relaxation at the weekend, and return home at night. In addition, (d) shows that people are keen to move to indoor areas instead of the outdoor park during storms. These observations demonstrate that regions with different semantics present different flow distributions in the presence of different external factors. Moreover, these external factors can compound and thus influence the actual distribution in complicated ways.

Inspired by techniques from the domain of image recovery, in the preliminary work we attack the FUFI problem by designing a neural network-based model entitled **Urban** Flow **M**agnifier (UrbanFM) [1] which resolves the above challenges with an innovative network structure. Firstly, we extract features from coarse-grained inputs using Convolutional Neural Networks (CNN) and perform upsampling based on high-level features. But in contrast to image processing, where the direct output is the target fine-grained image, we instead change the learning objective to the inference of the *distributions* of the fine-grained flows that capture how the flows in each superregion are distributed to their corresponding subregions. To this end, we present a *distributional upsampling* module with a novel and parameter-free layer entitled *N^2-Normalization* which provides superior performance over the image super-resolution baselines by exploiting the underlying structure of the FUFI problem. Moreover, we employ an external factor fusion subnet to capture the complexity of external impacts and produce a feature map that embeds the different impacts on different locations. Benefitting from the dedicated network architecture, UrbanFM outperforms all six baselines we have studied, including heuristics and state-of-the-art methods across all three evaluation metrics we have considered.

Nevertheless, we identify several limitations of UrbanFM as a preliminary work. 1) One major problem is that it was evaluated only for 4x upsampling. When the required upsampling scale becomes larger (e.g., 8x), UrbanFM can be limited as it performs upsampling in a *single forward pass*. Such a simple strategy increases the difficulty for the feature extraction layers when the upsampled space becomes much larger. Inspired by the concept of Pyramid structure [7], in this paper, we present an enhanced model named **Urban Py**ramid Network (UrbanPy) which inherits key advantages from UrbanFM while performing progressive upsampling instead of single-passing. 2) UrbanFM utilizes classic convolution layers with shared kernels for all locations, which however ignores the localized features from each superregion. To address this problem, we employ non-shared convolution layers and incorporate geographic information to enhance customization for each region. 3) The loss function employed by UrbanFM has not incorporated the structural nature of the problem. Instead, we introduce a KL-divergence loss for each region to bridge the gap between the problem nature and the learning objective.

To summarize, we make the following contributions:

- We formalize the problem of Fine-grained Urban Flow Inference, which is critical for modern urban information infrastructure construction. We show that the essence of this problem is to uncover the distributions of superregions over their associative subregions.
- We present UrbanFM, which exploits the problem structure and shows superior performance versus the baseline methods. Moreover, we identify the limitation of this preliminary work for large scale upsampling and present the improved method UrbanPy, which incorporates multiple key innovations over UrbanFM.
- We conducted extensive experiments using real-world datasets, including city-scale (i.e., Beijing) and district-scale (i.e. HappyValley, a theme park). Empirical results demonstrate the superiority of our methods compared to multiple state-of-the-art approaches.

2 Problem Formulation

This section first defines some notation and then formulates the problem of Fine-grained Urban Flow Inference (FUFI).

Definition 1 (Region) As shown in Figure 1, we partition an area of interest (e.g., a city) evenly into an $I \times J$ grid map based on longitude and latitude, where a grid elelment denotes a region [8]. Partitioning the city into smaller regions (i.e., using larger I, J) allows ones to obtain flow data with more detail, which produces a more fine-grained flow map.

Definition 2 (Flow Map) Let $\mathbf{X} \in \mathbb{R}_+^{I\times J}$ represent a flow map of a particular time, where each entry $x_{i,j} \in \mathbb{R}_+$ denotes the stay-in flow volume of the flow agents (e.g., vehicle, people, etc.) in region (i,j).

Definition 3 (Superregion & Subregion) In our FUFI problem, a coarse-grained grid map indicates the data granularity we can observe upon sensor reduction. It is obtained by combining nearby grids within an N-by-N range of a fine-grained grid map using a scaling factor N. Figure 1 illustrates an example when $N = 2$. Each coarse-grained grid in Figure 1(a) is composed of 2×2 smaller grids from Figure 1(b). We define the aggregated larger grid as a *superregion*, and its constituent smaller regions as *subregions*. Note that in this setting, superregions do not share subregions. Hence, the relationship between superregions and the corresponding subregions induces a special *structural constraint* in FUFI.

Definition 4 (Structural Constraint) The flow volume $x^c_{i,j}$ in a superregion of the coarse-grained grid map and the flows $x^f_{i',j'}$ in the corresponding subregions of the fine-grained counterpart obey the following equation:

$$x^c_{i,j} = \sum_{i',j'} x^f_{i',j'} \quad s.t. \lfloor \frac{i'}{N} \rfloor = i, \lfloor \frac{j'}{N} \rfloor = j. \tag{1}$$

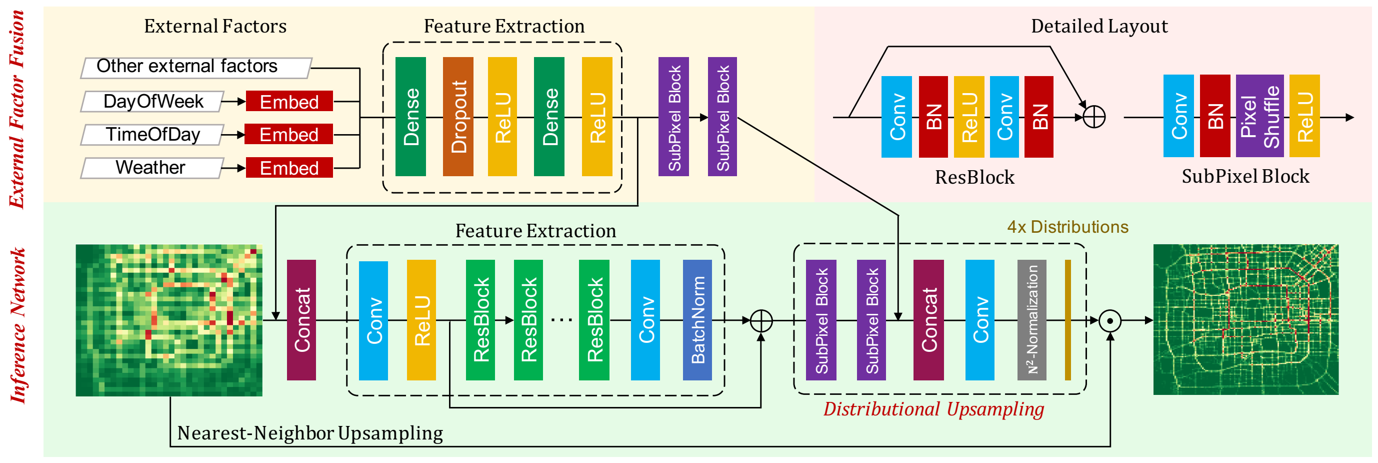


Fig. 3. The UrbanFM framework for $4\times$ upscaling ($N=4$). $\oplus$ denotes addition and $\odot$ denotes Hadamard product. Note that our framework allows other integer upscaling factor, not limited to power of 2.

For simplicity, $i = 1, 2, \dots, I$ and $j = 1, 2, \dots, J$ in our paper unless otherwise specified.

Problem Statement (Fine-grained Urban Flow Inference) Given an upscaling factor $N \in Z_+$ and a coarse-grained flow map $\mathbf{X}^c \in \mathbb{R}_+^{I\times J}$, infer the fine-grained counterpart $\mathbf{X}^f \in \mathbb{R}_+^{NI\times NJ}$ as accurately as possible subject to the structural constraints.

3 Urban Flow Magnifier

Figure 3 depicts the framework of UrbanFM which consists of two main components for conducting structurally constrained inference of fine-grained flows and capturing complex external influence on the flows, respectively. The inference network takes the coarse-grained flow map $\mathbf{X}^c$ as input, and then extract high-level features across the whole area by leveraging deep residual networks [9]. Taking extracted features as a priori knowledge, the *distributional upsampling* module outputs a flow distribution over the subregions of each superregion by introducing a dedicated N^2-*Normalization* layer. Finally, the Hadamard product of the inferred distribution with the original coarse-grained flow map gives the fine-grained flow map $\tilde{\mathbf{X}}^f$ as the network output. In an external factor fusion branch, we leverage embeddings and a dense network to extract pixel-wise external features at both coarse and fine granularity. The integration of external and flow features enables UrbanFM to exhibit fine-grained flow inference more effectively. In this section, we describe the designs of the two components, as well as the optimization scheme used in network training.

3.1 Inference Network

The inference network aims to produce a map of fine-grained flow distributions over subregions from a coarse-grained input. We follow the general procedure in image super-resolution (SR) methods, which is composed of two phases: 1) feature extraction; 2) inference upon upsampled features.

3.1.1 Feature Extraction

In the input stage, we use a convolutional layer (with 9×9 filter size and filter size F) to extract low-level features from the given coarse-grained flow map $\mathbf{X}^c$, and perform the first stage of fusion if external features are provided. Then M residual blocks with identical layout take the (fused) low-level feature maps as input and construct high-level feature maps. The residual block layout, as shown on the top right of Figure 3, follows the guideline in Ledig et al. [10] which contains two convolutional layers (3×3, F) followed by a batch normalization layer [11], with an intermediate ReLU [12] function to introduce non-linearity.

Since we utilize a fully convolutional architecture, the reception field grows larger as we stack the network deeper. In other words, each pixel at the high-level feature map will be able to capture distant or even citywide dependencies. Moreover, we use another convolutional layer (3×3, F) followed by batch normalization to guarantee non-trivial feature extraction. Finally, drawing from the intuition that the output flow distribution exhibits region-to-region dependencies on the original $\mathbf{X}^c$, we employ a skip connection to introduce identity mapping [13] between the low-level features and high-level features, building an information highway skipping over the residual blocks to allow efficient gradient back-propagation.

3.1.2 Distributional Upsampling

In the second phase, the extracted features first go through n sub-pixel blocks to perform an $N = 2^n$ upscaling operation which produces a hidden feature $\mathbf{H}^f \in \mathbb{R}^{F\times NI\times NJ}$. The sub-pixel block, as illustrated in Figure 3, leverages a convolutional layer (3×3, $F\times 2^2$) followed by batch normalization to extract features. Then it uses a PixelShuffle layer [14] to rearrange and upsample the feature maps to $2\times$ size and applies a ReLU activation at the end. After processing each sub-pixel block, the output feature maps are 2 times larger spatially with the number of channels unchanged. A convolutional layer (9×9, F_o) is applied post-upsampling, which maps $\mathbf{H}^f$ to a tensor $\mathbf{H}_o^f \in \mathbb{R}^{F_o\times NI\times NJ}$. $F_o = 1$ in our case for simplicity. In SR tasks, $\mathbf{H}_o^f$ is usually the final output for the recovered image with super-resolution. However, the structural constraint that is essential to FUFI has not yet been considered.

In order to impose the structural constraint on the network, one straightforward manner is to add a *structural loss*

Algorithm 1: N^2-Normalization

```
Input: x, scale_factor, $\epsilon$
Output: out
// x: an input feature map
// scale_factor: the upscaling factor
// $\epsilon$: a small number for numerical
   stability
// out: the structural distributions

sum = SumPooling(x, scale_factor);
sum = NearestNeighborUpsampling(sum,
  scale_factor);
out = x $\oslash$ (sum+$\epsilon$) // element wise division
```

L_s as a regularization term to the loss function:

$$L_s = \sum_{i,j} \left\| x^c_{i,j} - \sum_{i',j'} \tilde{x}^f_{i',j'} \right\|_F \quad s.t. \lfloor \frac{i'}{N} \rfloor = i, \lfloor \frac{j'}{N} \rfloor = j. \tag{2}$$

However, simply applying L_s does not improve the model performance, as we demonstrate in Section 5. Instead, we design an N^2-*Normalization* layer, which outputs a *distributions* over every patch of N-by-N subregions of an associated superregion. To achieve this, we reformulate Equation 1 as in the following:

$$\begin{aligned} x^c_{i,j} &= \sum_{i',j'} \alpha_{i',j'} x^c_{i,j} \\ s.t. \sum \alpha_{i',j'} &= 1,\ \alpha \in \mathbb{R}_+, \lfloor \frac{i'}{N} \rfloor = i, \lfloor \frac{j'}{N} \rfloor = j. \end{aligned} \tag{3}$$

The flow volume in each subregion is now expressed as a *fraction* of that in the superregion, i.e., $x^f_{i',j'} = \alpha_{i'j'} x^c_{i,j}$. We thus can treat the fraction as a probability. This allows us to interpret the network output in a meaningful way: the value in each subregion pixel states how likely the overall superregion flow will be allocated to the subregion (i', j'). With this reformulation, we shift our focus from directly generating the fine-grained flow to generating the flow distributions $\mathbf{D}'$. This essentially changes the network learning target and thus diverges from the traditional SR literature. To this end, we present the N^2-*Normalization* layer: N^2-*Normalization*$(\mathbf{H^f_o}) = \mathbf{D}'$, such that

$$\mathbf{D}'_{(i,j)} = \mathbf{H}^f_{o,(i,j)} / \sum_{\substack{i'=(\lfloor i/N \rfloor -1)*N+1, \\ j'=(\lfloor j/N \rfloor -1)*N+1}}^{\substack{i'=\lfloor i/N \rfloor *N, \\ j'=\lfloor j/N \rfloor *N}} \mathbf{H}^f_{o,(i',j')} \tag{4}$$

The N^2-Normalization layer induces no extra parameters to the network. Moreover, it can be easily implemented within a few lines of code (see Algorithm 1). Also, the operations can be fully paralleled and automatically differentiated at runtime. Remarkably, this reformulation relieves the network from concerning varying output scales and enables it to focus on producing a probability within $[0, 1]$ constraint.

Finally, we upscale $\mathbf{X}^c$ using nearest-neighbor upsampling [15] (`NN_Upsample`) with scaling factor N as the initial interpolation, and then generate the fine-grained inference by

$$\tilde{\mathbf{X}}^f = \texttt{NN\_Upsample}(\mathbf{X}^c; N) \odot \mathbf{D}'. \tag{5}$$

3.2 External Factor Fusion

External factors, such as weather, can have a complicated and vital influence on the flow distribution over the subregions. For instance, even if the total population in a city remains stable over time, during stormy weather people tend to move from outdoor regions to indoor regions. When different external factors entangle, the actual impact on the flow becomes more difficult to capture. Therefore, we design a subnet to handle external factors *all at once*.

In particular, we first separate the available external factors into two groups, continuous features, and categorical features. Continuous features including temperature and wind speed are directly concatenated to form a vector $\mathbf{e}_{con}$. As shown in Figure 3, categorical features include the day of week, the time of the day, and kind of weather (e.g, sunny, rainy). Inspired by previous studies [16], we transform the categorical attributes into low-dimensional vectors by feeding them into separate embedding layers, and then concatenate those embeddings to construct the categorical vector $\mathbf{e}_{cat}$. Then, the concatenation of $\mathbf{e}_{con}$ and $\mathbf{e}_{cat}$ gives the final external embedding, with $\mathbf{e} = [\mathbf{e}_{con}; \mathbf{e}_{cat}]$.

Once we obtain the concatenation vector $\mathbf{e}$, we feed it into a feature extraction module whose structure is depicted in Figure 3. By using dense layers, the different external factors are compounded to construct a hidden representation, which models their complicated interaction. The module provides two outputs: the coarse-grained feature maps $\mathbf{H}^c_e$ and the fine-grained feature maps $\mathbf{H}^f_e$, where $\mathbf{H}^f_e$ is obtained by passing $\mathbf{X}^c_e$ through n sub-pixel blocks similar to the ones in the inference network. Intuitively, $\mathbf{H}^c_e$ (respectively $\mathbf{H}^f_e$) is the spatial encoding for $\mathbf{e}$ in the coarse-grained (fine-grained) setting, modeling how each superregion (subregion) individually responds to the external factors. Therefore we concatenate $\mathbf{H}^c_e$ with $\mathbf{X}^c$, and $\mathbf{H}^f_e$ with $\mathbf{H}^f$ in the inference network. The early fusion of $\mathbf{H}^c_e$ and $\mathbf{X}^c$ allows the network to learn to extract a high-level feature describing not only the citywide flow, but also the external factors. In addition, the fine-grained $\mathbf{H}^f_e$ carries the external information all the way to the rear of the inference network, playing a similar role as an information highway, and thus prevents information perishing in the deep network.

3.3 Loss Function

UrbanFM provides an end-to-end mapping from coarse-grained input to fine-grained output, which is differentiable everywhere. Therefore, we can train the network through auto back-propagation, by providing training pairs $(\mathbf{X}^c, \mathbf{X}^f)$ and calculating empirical loss between $(\mathbf{X}^f, \tilde{\mathbf{X}}^f)$, where $\mathbf{X}^f$ is the ground truth and $\tilde{\mathbf{X}}^f$ is the outcome inferred by our network. As pixel-wise Mean Square Error (MSE) is a widely used cost function in many SR tasks, we employ the same in this work as follows:

$$L_{MSE}(\mathbf{X}^f, \tilde{\mathbf{X}}^f; \mathbf{\Theta}) = \|\mathbf{X}^f - \tilde{\mathbf{X}}^f\|^2_F \tag{6}$$

where $\mathbf{\Theta}$ denotes the set of parameters in UrbanFM.

4 Urban Pyramid Network

The design of UrbanFM has followed two key principles: first, reconstruct a fine-grained flow map according to high-level features; second, embed the structural constraint in

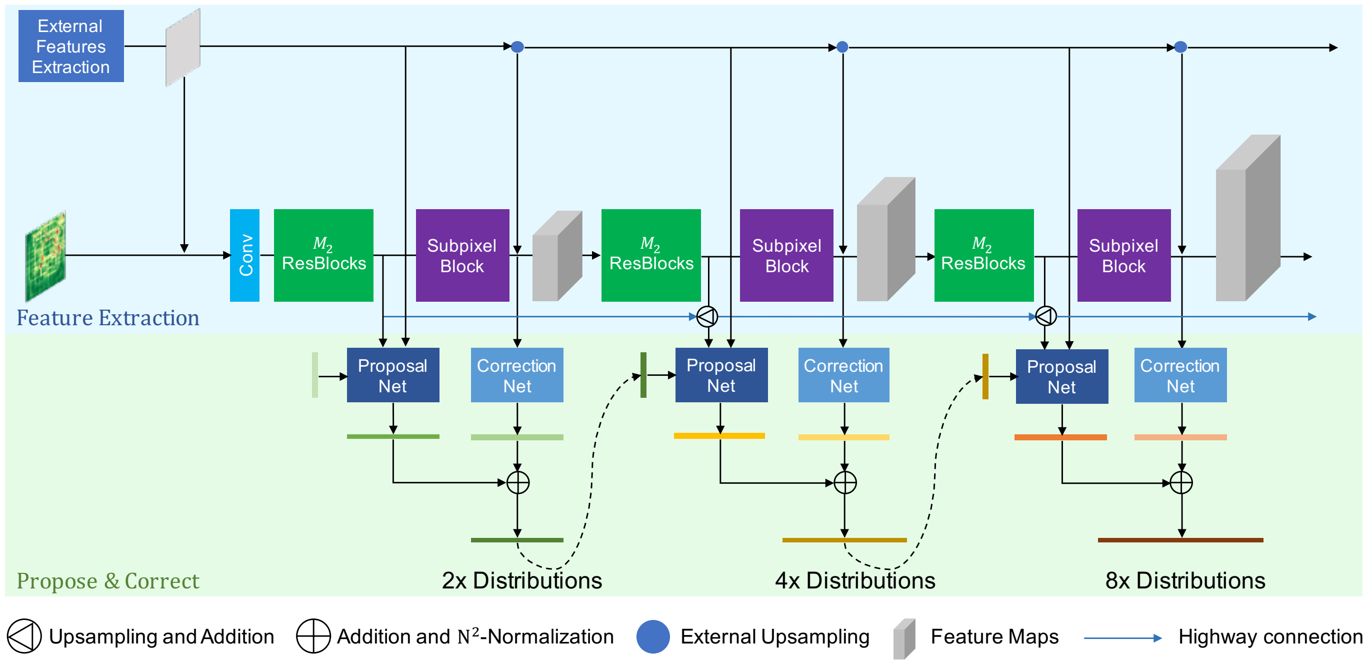


Fig. 4. A UrbanPy framework for $2\times$, $4\times$ and $8\times$ upscaling. We employ a cascading strategy to progressively upsample the coarse-grained inputs. At each level, we first prepare features from the input flow map and external factors. Then a propose-and-correct component is used to cooperatively produce two views of the target distribution map. We aggregate and renormalize the two views using N^2-Normalization to give the map of mixture distributions which will then be an input for the next level. The foremost distribution map $\mathbf{D}_0$ is initialized with an all-ones matrix. Note that we omit nearest neighbor upsampling and the Hadamard product to avoid the redundant presentation of elements shown in Figure 3.

the model design. Maintaining those two principles, we now present UrbanPy which advances the UrbanFM framework by resolving three limitations: 1) a single upsampling process, 2) non-distinguishable features and 3) inconsistent MSE loss. The overall architecture is depicted in Figure 4.

4.1 Pyramid Architecture

Upsampling for a large scale (e.g., 16x) in a single step as in UrbanFM is very challenging. Therefore, UrbanPy decomposes the overall upsampling objective into multiple subprocesses. For objective decomposition, UrbanPy employs a pyramid architecture consists of multiple components, where each component functions as an atomic upsampler for a small scale (e.g. 2x). Such decomposition allows the network to divide a difficult task into much easier subtasks such that each component can solve its own subtask more effectively.

Specifically, taking a coarse-grained flow map as input, our model decomposes the high-scale upsampling task into L consecutive upsampling subtasks with upsampling factors $[s_1, s_2, \ldots, s_l, \ldots s_L]$ respectively, where $s_L = N$.[1] In accordance with the decomposition, UrbanPy employs L components of similar structure where each component upsamples from s_l to s_{l+1}. The common component structure includes two modules: 1) feature extraction and 2) propose-and-correct. The feature extraction module abstracts a higher-level representation of the original input and transforms the feature maps from a lower scale to a higher scale. Then the processed representations are fed into a proposal network and a correction network, which collaboratively infer the flow distributions at the l^{th} scale. We describe the two modules further in the following.

1. For simplicity, upsample rates for the width and height are described as being the same here but need not be in the general case.

4.1.1 *Feature Extraction*

The input to the l^{th} feature extraction component a feature tensor $\mathbf{H}_{l-1} \in \mathcal{R}^{s_{l-1}I \times s_{l-1}J \times F_{l-1}}$ generated from the previous feature extraction component. Given $\mathbf{H}$, we first uses M_2 number of residual blocks of the same layout to construct a high-level representation $\tilde{\mathbf{H}}_l \in \mathcal{R}^{s_{l-1}I \times s_{l-1}J \times F_l}$ without changing the spatial dimensions. In order to meet the upscaled dimension at the current level, we employ a Subpixel block to enlarge the spatial dimension of $\tilde{\mathbf{H}}_l$ and produce upsampled feature maps $\mathbf{H}_l \in \mathcal{R}^{s_l I \times s_l J \times F_l}$. $\tilde{\mathbf{H}}_l$ and $\mathbf{H}_l$ thus provide two different views of the inputs, and are used as separate features for the following proposal network and correction network respectively. For simplicity, we set $F_l = F$ for $l = 1, \ldots, L$.

Highway Connection. Larger upscaling rates require stacking more feature extraction component, leading to a deeper network architecture. Although we have utilized residual blocks to facilitate gradient passage during back-propagation, the existence of other layers (e.g., upsampling) can affect the dynamics such that the deeper network becomes harder to train. Therefore, we add highway connections (denoted as blue arrows in Figure 4) across components such that previous features can be directly reused in the deeper layers. Specifically, given a list of previous representations $[\tilde{\mathbf{H}}_1, \ldots, \tilde{\mathbf{H}}_{l-1}]$, we upsample them to the scale of s_l and then aggregate with the current representation $\tilde{\mathbf{H}}_l$ by addition, which gives $\tilde{\mathbf{H}}_l^*$.

External Factor Fusion. We employ the same external factor fusion module used in UrbanFM for extracting the external features. Let $\mathbf{H}_e^1$ be $\mathbf{H}_e^c$. At each level we recruit a subpixel block to upsample $\mathbf{H}_e^{l-1}$ to $\mathbf{H}_e^l$. We use the same weights in each upsampler to reduce the number of parameters as we observed no obvious advantage of using distinct weights in our experiments.

4.1.2 Propose and Correct

Inspired by the idea of Laplacian Pyramid (LP), where an interpolated blurred image and its difference to the original image is modeled, we present a Propose-and-Correct (PC) architecture such that the proposal network aims to generate a prototype distribution map, and the correction network is responsible for correcting the prototype. Both LP and PC share a similar nature of recruiting multi-view information source for inferring the target, which allows to further improve the inference performance.

Specifically, given representations $\tilde{\mathbf{H}}_l^*$ and $\mathbf{H}_e^{l-1}$, the proposal network produces a prototype $\tilde{\mathbf{D}}_l \in \mathcal{R}^{s_l I \times s_l J}$ by embracing the flow distributions $\mathbf{D}_{l-1} \in \mathcal{R}^{s_{l-1} I \times s_{l-1} J}$ produced at the previous level (see the dotted lines in Figure 4). Note that these connections give the whole network a cascade structure, which improves the consistency between different levels as there are obvious correlations between flows at different granularities. The interior structure of the proposal network is depicted in Figure 5(a), where we stack R number of residual blocks to strengthen the capacity of the proposal network, followed by an N^2-Normalization layer to enforces the hierarchical structure of the output distributions at the current scale. The dotted elements are described in the Section 4.2.

Taking $\mathbf{H}_l$ and $\mathbf{H}_e^l$ as input, the correction network employs a similar structure as the proposal network (i.e. convolutional layers followed by N^2-Normalization) to generate a correction distribution map $\hat{\mathbf{D}}_l \in \mathcal{R}^{s_l I \times s_l J}$. But the correction network is designed to be light-weight. One reason is that adjusting the distribution based on the proposal network is a simpler task. Moreover, the correction network actually can take advantage of the weights of the upsample block from the feature extraction branch whose output also serves as direct input to the correction network. Therefore, we design the correction network with just one convolution layer to transform the upsampled feature maps.

Finally, we add the prototype $\tilde{\mathbf{D}}_l$ and correction $\hat{\mathbf{D}}_l$ distribution maps in an element-wise manner, and then renormalize the results to generate the final distribution map $\mathbf{D}_l$. That is,

$$\mathbf{D}_l' = \text{N}^2\texttt{-Normalization}(\tilde{\mathbf{D}}_l + \hat{\mathbf{D}}_l). \tag{7}$$

Akin to UrbanFM, the generated distribution map is Hadamard-multiplied by the interpolated coarse-grained input to give the inferred fine-grained flow prediction at the $s_l \times$ scale.

A Probabilistic View. Note that the PC architecture is different from the LP used in image superresolution [7, 17, 18]. The latter *adds* the interpolated image with the predicted residual image to obtain the final inference. However, in the problem of FUFI we are concerned with modeling flow distributions, but distributions do not exhibit *closure* under addition (i.e., $p_1 + p_2 \notin \mathcal{P}$ where $p_1, p_2 \in \mathcal{P}$ and $\mathcal{P}$ is the set of distributions). Instead, UrbanPy infers the final distribution map as a symmetric mixture density of the prototype and the correction, where each super-region is represented by a mixture probability distribution.

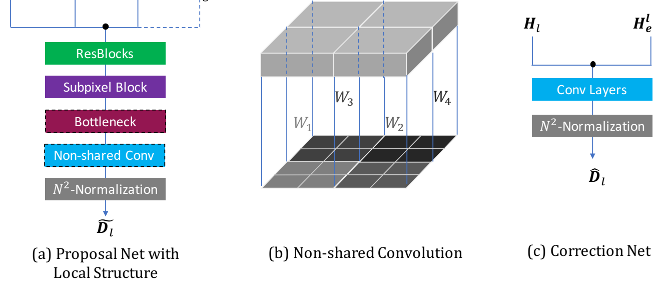


Fig. 5. Structure of the correction net, the proposal net and the non-shared convolution layer. (a) and (c) uses black dots to denote concatenation; $\mathbf{H}_g$ is the extra geographic feature for obtaining local structure. (b) uses different color of W to denotes different kernel weights; similar to urbanFM, it outputs a feature map of channel size F_o.

4.2 Local Structure

Each super-region in the coarse-grained map can cover a very large area. E.g., each grid in the 8-by-8 granularity contains about 6.25 km^2 area in Beijing. The region-specific properties (building layout, road plan) of each grid can vary significantly from one grid to another, and thus have different impacts on people's flow pattern. To capture such local property, we include the geographic features as additional knowledge and employ a non-shared convolution layer to allow customized feature extraction for each super-region.

4.2.1 Geographic Embeddings

For each level s_l, we obtain level-specific geographic features including POI and road network. For POI, we obtain a set of POI density maps of the city across different categories (e.g., education and entertainment), which results in (raw) feature maps $\mathbf{H}_{poi} \in \mathcal{R}^{s_l I \times s_l J \times C_{poi}}$, where C_{poi} denotes the total number of POI categories. Likewise, in terms of the road network structure, we obtain the tier-1, tier-2 and tier-3 road density for each region and construct a road network feature tensor $\mathbf{H}_{rn} \in \mathcal{R}^{s_l I \times s_l J \times 3}$. Then the geographic feature $\tilde{\mathbf{H}}_g = [\mathbf{H}_{poi}; \mathbf{H}_{rn}]$ is given by concatenation of both along the last dimension. $\tilde{\mathbf{H}}_g$, however, is very sparse. To mitigate the sparsity, we perform feature compression by pre-training a Stacked Denoising Auto-encoder [19] and then use the hidden code $\mathbf{H}_g = \texttt{encoder}(\tilde{\mathbf{H}}_g) \in \mathcal{R}^{s_l I \times s_l J \times C_g}$ as a compressed knowledge of the raw features. Eventually, each grid can be represented by an individual embedding $\mathbf{h}_{i,j}^g \in \mathbf{H}_g$ of size N_g.

4.2.2 Non-shared Convolution

The UrbanFM model uses the classic convolution layers such that the kernel weights are shared globally, which can be insufficient to capture the peculiarity of each superregion. Therefore, besides recruiting geographic features, we use separated weights to produce a more local representation of each super-region before applying N^2-Normalization, as is shown in Figure 5.

First, we define the classic discrete convolution operation $*$ according to the formulation from [20] as follows[2]. Let $F : \mathbb{Z}^2 \rightarrow \mathcal{R}$ be a discrete function. Let $\Omega_r = [1-r, r]^2 \cap \mathbb{Z}^2$

2. To keep consistent, we use even kernel width here. An odd kernel width simply modifies $\Omega_r = [-r, r]^2 \cap \mathbb{Z}^2$.

and let $k : \Omega_r \rightarrow \mathcal{R}$ be a discrete filter of size $(2r)^2$. Then the $*$ is defined as:

$$(F * k)(\mathbf{p}) = \sum_{\mathbf{s}+\mathbf{t}=\mathbf{p}} F(\mathbf{s})k(\mathbf{t}). \tag{8}$$

To generalize the classic convolution, we introduce a set of kernels $\mathcal{K} = \{k_{i,j}\}$ where $(i,j) \in \mathbb{Z}^2$ and define the local non-shared convolution $*'$ as:

$$(F *' \mathcal{K})(\mathbf{p}) = \sum_{\substack{\mathbf{s}+\mathbf{t}=\mathbf{p},\\ \mathbf{p}=(r+2ri, r+2rj)}} F(\mathbf{s})k_{i,j}(\mathbf{t}). \tag{9}$$

The kernel width $2r$ is thus equal to the convolution stride. Specifically, in each level l, we set $2r$=s_l to force each kernel to focus on a specific super-region as shown in Figure 5(c). To reduce the parameter cost for customization, a bottleneck layer [9] is deployed in advance to compress the channel of the feature maps provided by previous layers to 2.

4.3 Distributional Loss

UrbanFM measures the MSE between the ground truth flow and the predicted *flow values*, and uses it as the loss to optimize the network parameters. Such loss is straight forward, but ignores the underlying structure of the problem. To resolve such inconsistency, we step deeper into the problem by directly measure the divergence existing between the truth distribution and the predicted *flow distributions*.

In particular, in level l, we first prepare the ground truth $\mathbf{D}_l$=$\mathbf{X}_l/\texttt{NN\_Upsample}(\mathbf{X}_1; s_l)$, where $\texttt{NN\_Upsample}(\cdot; s_l)$ indicates nearest neighbour upsampling by scale s_l. Once we obtain the inferred distribution map $\mathbf{D}'_l = [\mathbf{d}'_1, \dots, \mathbf{d}'_{I\times J}]$ that contains $I \times J$ different distributions, the total distributional loss L_D between the two set of distributions is computed via KL-divergence:

$$L^l_D(\mathbf{D}'_l, \mathbf{D}_l; \mathbf{\Theta}) = \sum_{i,j=1,1}^{I,J} KL(\mathbf{d}'_{i,j}, \mathbf{d}_{i,j})$$
$$where \quad KL(\mathbf{d}', \mathbf{d}) = \sum_{p=1}^{s_l^2} d'_p \log \frac{d'_p}{d_p}.$$

The distributional loss exploits the essence of the model and is defined on each superregion-subregion distilling. Using L_D alone can train the network till convergence, but its asymmetry can produce unstable gradients which slows down the training process [21]. Therefore, we combine both the MSE loss and distributional loss across at each level, and aggregate through all levels to constitute the overall loss function:

$$L = \textstyle\sum_l \alpha L^l_D + (1-\alpha)L^l_{MSE}, \tag{10}$$

where α is the coefficient to control the scale of the two losses. In experiments, we set α=1e-2 by default to balance the magnitude of the gradients from the two losses, giving stable multi-task training according to [22].

5 Experiments

Our experiments aim to quantitively and qualitatively examine the capacity of the presented two models in a citywide scenario. Therefore, we conduct extensive experiments using taxi flows in Beijing to comprehensively test the model from different aspects. Different from the preliminary evaluations that focus only on $4\times$ upsampling, we conduct experiments involving four different scales. In addition to citywide, we conduct further experiments in a theme park, namely Happy Valley, to show the adaptivity of our models in a relatively small area.

5.1 Experimental Settings

5.1.1 Datasets

Table 1 details two real-world datasets, TaxiBj and HappyValley, where each dataset contains two sub-datasets: urban flows and external factors. Since a number of fine-grained flow data are available as ground truth, in this paper, we can obtain the a coarse-grained flows by aggregating subregion flows from the fine-grained counterparts. As our empirical evaluations span across multiple scales, we obtain data at each granularity separately. When conducting experiments for $s_l\times$ upscaling, we aggregate the subregions in a $s_l \times s_l$ area to generate the flows of the corresponding superregion.

- **TaxiBJ**[3]: This dataset indicates the taxi flows traveling throughout Beijing. Figure 1 gives an example when the studied area is split into 32×32 grids. where each grid reports the coarse-grained flow information every 30 minutes within four different periods: P1 to P4 (detailed in Table 1). In our experiments, we partition the data into non-overlapping training, validation and test data by a ratio of 2:1:1 respectively for each period. For example, in P1 (7/1/2013-10/31/2013), we use the first two-month data as the training set, the next month as the validation set, and the last month as the test set. With this dataset, we construct coarse-grained data of 4 different granularities (i.e., 8×8, 16×16, 32×32 and 64×64) as the coarse-grained inputs, targeting upsampling factor $N = 16, 8, 4, 2$ respectively. The data partition for each granularity is the same.
- **HappyValley**: We obtain this dataset by crawling from an open website[4] which provides hourly gridded crowd flow observations for a theme park named Beijing Happy Valley, with a total $5\times10^5 m^2$ area coverage, from 1/1/2018 to 10/31/2018. As shown in Figure 6, we partition this area with 25×50 uniform grids in coarse-grained setting, and target a fine granularity at 50×100 with an upscaling factor $N = 2$. Note that in this dataset, one special external factor is the ticket price, including day price and night price, obtained from the official account of HappyValley in WeChat. Regarding the smaller area, crowd flows exhibit large variance across samples given the 1-hour sampling rate. Thus, we use a ratio of 8:1:1 to split training, validation, and test set to provide more training data.

5.1.2 Evaluation Metrics

We use three common metrics for urban flow data to evaluate the model performance from different facets. Specifically, Root Mean Square Error (RMSE) is defined as:

$$RMSE = \sqrt{\frac{1}{z}\sum_{s=1}^{z} \left\| \mathbf{X}^f_s - \tilde{\mathbf{X}}^f_s \right\|^2_F},$$

3. See our GitHub https://github.com/yoshall/UrbanFM
4. heat.qq.com

TABLE 1
Dataset Description.

Dataset	TaxiBJ	HappyValley
Time span	P1: 7/1/2013-10/31/2013 P2: 2/1/2014-6/30/2014 P3: 3/1/2015-6/30/2015 P4: 11/1/2015-3/31/2016	1/1/2018-10/31/2018
Time interval	30 minutes	1 hour
Coarse-grained size	32×32	25×50
Fine-grained size	128×128	50×100
Upscaling factor (N)	4	2
External factors (meteorology, time and event)		
Weather (e.g., Sunny)	16 types	8 types
Temperature/°C	[-24.6, 41.0]	[-15.0, 39.0]
Wind speed/mph	[0, 48.6]	[0.1, 15.5]
# Holidays	41	33
Ticket prize/¥	/	[29.9, 260]
Geographic features		
Road network	density	/
Point of Interest	density	/

where z is the total number of samples, $\tilde{\mathbf{X}}^f_s$ is s-th the inferred value and $\mathbf{X}^f_s$ is corresponding ground truth. Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) are defined as: $MAE = \frac{1}{z}\sum_{s=1}^{z}\|\mathbf{X}^f_s - \tilde{\mathbf{X}}^f_s\|_{1,1}$ and $MAPE = \frac{1}{z}\sum_{s=1}^{z}\|(\mathbf{X}^f_s - \tilde{\mathbf{X}}^f_s) \oslash \mathbf{X}^f_s\|_{1,1}$. In general, RMSE favors spiky distributions, while MAE and MAPE focus more on the smoothness of the outcome. Smaller metric scores indicate better model performances.

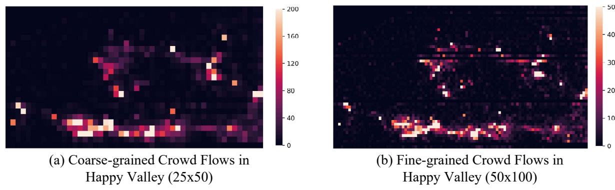


Fig. 6. Visualization of real crowd flows in HappyValley.

5.1.3 Baselines

We compare our proposed model with seven baselines that belong to the following three classes: (1) Heuristics, (2) Single-pass upsampling, and (3) Progressive upsampling. The heuristic methods are designed based on intuition or empirical knowledge. In the single-pass category, we include four state-of-the-art methods for single image super-resolution, the domain from which we are inspired to design UrbanFM. In the progressive upsampling branch, we involve two methods with different progressive strategies: stacking and cascading, where one is the state of the art on statistical upsampling for climate data and the other is for image super-resolution. We detail them as follows:

Heuristic methods:

- **Mean Partition (Mean)**: We evenly distribute the flow from each superregion in a coarse-grained flow map to the N^2 subregions, where N is the upscaling factor.
- **Historical Average (HA)**: Similar to distributional upsampling, HA treats the value over each subregion a fraction of the value in the respective superregion, where the fraction is computed by averaging all training data.

Single-pass methods:

- **SRCNN** [23]: SRCNN presented the first successful introduction of convolutional neural networks (CNNs) into the SR problems. It consists of three layers: patch extraction, non-linear mapping, and reconstruction. Filters of spatial sizes 9×9, 5×5, and 5×5 were used respectively. The number of filters in the two convolutional layers is 64 and 32 respectively. In SRCNN, the low-resolution input is upscaled to the high-resolution space using a single filter (commonly bicubic interpolation) before reconstruction.
- **ESPCN** [14]: Bicubic interpolation used in SRCNN is a special case of the deconvolutional layer. To overcome the low efficiency of such deconvolutional layer, Efficient Sub-Pixel Convolutional Neural Network (ESPCN) employs a sub-pixel convolutional layer aggregates the feature maps from LR space and builds the SR image in a single step.
- **VDSR** [24]: Since both SRCNN and ESPCN follow a three-stage architecture, they have several drawbacks such as slow convergence speed and limited representation ability. Inspired by the VGG-net, Kim et al. presents a Super-Resolution method using Very Deep neural networks with depth up to 20. This study suggests that a large depth is necessary for the task of SR.
- **SRResNet** [10]: SRResNet enhances VDSR by using the residual architecture presented by He et al.[9]. The residual architecture allows one to stack a much larger number of network layers, which bases many benchmark methods in computer vision tasks.

Progressive methods[5]:

- **DeepSD** [25]: DeepSD is the state-of-the-art method on statistical upscaling (i.e., super-resolution)for meteorological data. It employs the stacking strategy by independently training multiple SRCNNs, each aims at downscaling for an intermediate level. It performs further upsampling by simply stacking up those pretrained SRCNNs. This method, however, is slow as it needs to perform interpolation first and then extract features on the large-size feature maps. Another state-of-the-art DeepDPM [26] also employs this same technique for a different task.
- **LapSRN** [7]: LapSRN is named due to the Laplacian pyramid structure. At each level, it predicts a residual image and then adds with the interpolated output from the previous level to construct the current prediction. The training of LapSRN employs a cascading strategy such that the whole model is trained end to end.

5.1.4 Variants

We study the following variants of UrbanFM to evaluate the roles of different components.

- **UrbanFM-ne**: We simply remove the external factor fusion subnet from our method, which can help reveal the significance of this component.
- **UrbanFM-sl**: Upon removing the external subnet, we further replace the distributional upsampling module by using sub-pixel blocks and L_s to consider the structural constraint in this variant.

Study on the variants of UrbanPy involves choosing the different depth of the proposal network R, different filter

5. As stacking of UrbanFMs gives similar or worse results over LapSRN at large scales, we show the results for DeepSD and LapSRN.

TABLE 2
Qualititative results for model comparison. We conducted inference experiments for four different scales, where all target the same endpoint with 128×128 resolution. For each scale, the results for single-process and progressive upscaling are presented separately. Across all methods, we use $\underline{\mathbf{underlined\ bold}}$, **bold** and $\underline{\text{underline}}$ to indicate the best, the second best and the third performance, respectively. This helps us to identify the performance change along the enlarging of upscaling.

Methods	Upscales	P1			P2			P3			P4		
		RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE	RMSE	MAE	MAPE
MEAN	2	16.899	8.931	2.935	21.557	11.373	3.477	22.111	11.876	3.633	15.369	8.218	2.766
HA	2	3.494	$\underline{1.723}$	$\underline{0.306}$	3.932	1.933	$\underline{0.305}$	4.072	2.002	$\underline{0.299}$	3.063	1.547	$\underline{0.287}$
SRCNN	2	3.216	1.793	0.433	3.500	1.998	0.468	3.587	2.034	0.446	2.861	1.643	0.422
VDSR	2	3.203	1.750	0.387	3.523	1.894	0.325	3.575	1.965	0.369	2.776	1.533	0.337
ESPCN	2	3.170	1.789	0.433	3.472	1.969	0.432	3.564	2.014	0.419	2.813	1.608	0.398
SRResNet	2	$\underline{3.101}$	1.742	0.430	$\underline{3.383}$	$\underline{1.840}$	0.351	**3.481**	$\underline{1.889}$	0.336	**2.732**	$\underline{1.518}$	0.347
UrbanFM	2	$\underline{\mathbf{3.015}}$	$\underline{\mathbf{1.553}}$	$\underline{\mathbf{0.265}}$	$\underline{\mathbf{3.344}}$	$\underline{\mathbf{1.729}}$	$\underline{\mathbf{0.260}}$	$\underline{\mathbf{3.415}}$	$\underline{\mathbf{1.783}}$	$\underline{\mathbf{0.262}}$	$\underline{\mathbf{2.675}}$	$\underline{\mathbf{1.397}}$	$\underline{\mathbf{0.248}}$
DeepSD	2	3.216	1.793	0.433	3.500	1.998	0.468	3.587	2.034	0.446	2.861	1.643	0.422
LapSRN	2	3.202	1.763	0.396	3.468	1.900	0.370	$\underline{3.584}$	1.959	0.360	2.797	1.545	0.338
UrbanPy	2	**3.093**	**1.578**	**0.268**	**3.420**	**1.758**	**0.264**	$\underline{3.584}$	**1.843**	**0.268**	$\underline{2.759}$	**1.428**	**0.252**
MEAN	4	20.918	12.019	4.469	20.918	12.019	5.364	27.442	16.029	5.612	19.049	11.070	4.192
HA	4	4.741	2.251	0.336	5.381	2.551	0.334	5.594	2.674	$\underline{0.328}$	4.125	2.023	0.323
SRCNN	4	4.297	2.491	0.714	4.612	2.681	0.689	4.815	2.829	0.727	3.838	2.289	0.665
VDSR	4	4.159	2.213	0.467	4.586	2.498	0.486	4.730	2.548	0.461	3.654	1.978	0.411
ESPCN	4	4.206	2.497	0.732	4.569	2.727	0.732	4.744	2.862	0.773	3.728	2.228	0.711
SRResNet	4	4.164	2.457	0.713	4.524	2.660	0.688	4.690	2.775	0.717	3.667	2.189	0.637
UrbanFM	4	**3.991**	**2.036**	**0.331**	$\underline{4.374}$	$\underline{2.256}$	$\underline{\mathbf{0.322}}$	**4.539**	$\underline{2.348}$	$\underline{\mathbf{0.323}}$	**3.526**	**1.831**	$\underline{\mathbf{0.310}}$
DeepSD	4	4.156	2.368	0.614	4.554	2.612	0.621	4.692	2.739	0.682	3.877	2.297	0.652
LapSRN	4	$\underline{3.997}$	$\underline{2.040}$	$\underline{0.339}$	$\underline{\mathbf{4.353}}$	**2.235**	$\underline{0.324}$	**4.539**	**2.343**	0.330	$\underline{3.531}$	$\underline{1.841}$	$\underline{0.315}$
UrbanPy	4	$\underline{\mathbf{3.949}}$	$\underline{\mathbf{1.997}}$	$\underline{\mathbf{0.330}}$	**4.359**	$\underline{\mathbf{2.227}}$	**0.323**	$\underline{\mathbf{4.519}}$	$\underline{\mathbf{2.319}}$	**0.326**	$\underline{\mathbf{3.514}}$	$\underline{\mathbf{1.821}}$	**0.314**
MEAN	8	22.565	13.205	5.221	28.903	16.871	6.305	29.677	17.617	6.587	20.606	12.168	4.882
HA	8	5.629	2.682	0.442	6.429	3.058	0.443	6.717	3.211	0.431	4.959	2.433	0.423
SRCNN	8	6.103	3.433	1.027	6.569	3.708	0.971	6.959	4.012	1.086	5.518	3.181	0.935
VDSR	8	5.178	2.681	0.580	5.482	2.821	0.489	5.878	3.069	0.543	4.623	2.416	0.481
ESPCN	8	4.854	2.664	0.664	5.291	2.854	0.580	5.529	2.981	0.570	4.311	2.368	0.547
SRResNet	8	4.783	2.554	0.579	5.215	2.807	0.572	5.492	2.935	0.551	4.298	2.297	0.487
UrbanFM	8	**4.748**	$\underline{2.373}$	**0.377**	$\underline{5.224}$	$\underline{2.647}$	**0.366**	$\underline{5.488}$	$\underline{2.776}$	**0.358**	**4.195**	**2.130**	**0.342**
DeepSD	8	5.412	3.056	0.831	5.680	3.175	0.756	6.023	3.397	0.804	4.733	2.452	0.450
LapSRN	8	$\underline{4.772}$	**2.355**	$\underline{0.401}$	**5.197**	**2.578**	$\underline{0.375}$	**5.456**	**2.721**	$\underline{0.383}$	$\underline{4.209}$	$\underline{2.141}$	$\underline{0.369}$
UrbanPy	8	$\underline{\mathbf{4.572}}$	$\underline{\mathbf{2.237}}$	$\underline{\mathbf{0.352}}$	$\underline{\mathbf{5.003}}$	$\underline{\mathbf{2.476}}$	$\underline{\mathbf{0.339}}$	$\underline{\mathbf{5.259}}$	$\underline{\mathbf{2.610}}$	$\underline{\mathbf{0.342}}$	$\underline{\mathbf{4.071}}$	$\underline{\mathbf{2.052}}$	$\underline{\mathbf{0.336}}$
MEAN	16	23.157	13.622	5.540	29.695	17.390	6.722	30.521	18.165	7.035	21.193	12.554	5.191
HA	16	6.307	2.920	0.459	7.289	3.358	0.461	7.619	3.534	0.448	5.597	2.658	0.442
SRCNN	16	9.987	5.699	1.779	9.198	5.148	1.647	10.912	6.226	1.822	8.181	4.618	1.490
VDSR	16	6.313	2.994	0.551	6.780	3.216	0.468	7.074	3.429	0.517	5.758	2.867	0.572
ESPCN	16	5.373	2.914	0.762	5.956	3.211	0.750	6.256	3.422	0.806	4.892	2.699	0.715
SRResNet	16	5.381	2.672	0.504	5.959	3.056	0.594	6.295	3.189	0.559	5.066	2.596	0.551
UrbanFM	16	$\underline{5.344}$	$\underline{2.573}$	$\underline{0.383}$	$\underline{5.968}$	$\underline{2.898}$	$\underline{0.371}$	$\underline{6.241}$	$\underline{3.033}$	$\underline{0.366}$	**4.839**	$\underline{2.362}$	$\underline{0.368}$
DeepSD	16	5.983	3.223	0.811	6.392	3.380	0.725	6.818	3.644	0.787	5.400	2.671	0.441
LapSRN	16	**5.244**	**2.449**	**0.357**	**5.820**	**2.729**	$\underline{\mathbf{0.339}}$	**6.104**	**2.879**	**0.343**	$\underline{4.860}$	**2.321**	**0.346**
UrbanPy	16	$\underline{\mathbf{5.204}}$	$\underline{\mathbf{2.434}}$	$\underline{\mathbf{0.357}}$	$\underline{\mathbf{5.750}}$	$\underline{\mathbf{2.719}}$	**0.352**	$\underline{\mathbf{6.061}}$	$\underline{\mathbf{2.868}}$	$\underline{\mathbf{0.346}}$	$\underline{\mathbf{4.778}}$	$\underline{\mathbf{2.290}}$	**0.350**

TABLE 3
P-value of Wilcoxon signed-rank test.

Test Group	RMSE	MAE	MAPE
UrbanFM-SRResNet	5.6e-4	2.2e-4	2.2e-4
UrbanPy-LapSRN	4.0e-4	2.2e-4	4.5e-3
UrbanPy-UrbanFM	3.5e-2	6.1e-3	3.9e-2

depth M_2, and filter size F for each feature extraction module at each level. We denote the variants by M_2-F-R and omit the name as shown in Table 4 for a more succinct presentation.

5.1.5 Training Details & Hyperparameters

Our model, as well as the baselines, are completely implemented by PyTorch with one TITAN V GPU. We use Adam [27], an algorithm for stochastic gradient descent, to perform network training with learning rate lr=1e-4 and use batch size being 16 for the single-pass methods. We also apply a staircase-like schedule by halving the learning rate every 20 epochs. In the external subnet, there are 128 hidden units in the first dense layer with dropout rate 0.3, and $I \times J$ hidden units in the second dense layer. We embed DayOfWeek to $\mathbb{R}^2$, HourOfDay to $\mathbb{R}^3$, and weather condition to $\mathbb{R}^3$. Besides, for VDSR and SRResNet, we use the default settings in their paper. Since SRCNN, ESPCN performs poorly with default settings, we test different hyperparameters for them and finally use 384 as the number of filters in their two convolutional layers.

For progressive methods, we stack pre-trained SRCNN with 384 filter size to construct DeepSD, and default hyperparameters for LapSRN. For UrbanPy, we use D=8 for 2×

tasks to endow more power for the network and D=4 for other scales unless otherwise specified. We early stop our training according to the validation results and if the model performance is not altered after 50 epochs.

5.2 Results on TaxiBJ

5.2.1 Model Comparison

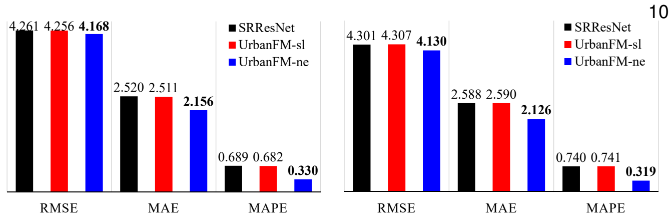
This subsection compares the model effectiveness against the baselines. We report the results of UrbanFM with M-F being 16-64 and UrbanPy with M_2-F-R being 4-64-4 as our default settings. Further experiments on variants regarding different M-F and M_2-F-R will be discussed later. Table 2 illustrates the overall performances of all methods for the TaxiBJ dataset for tasks with 2×, 4×, 8× and 16× upscaling. Due to space limitation, the key tests of significance regarding the results of this table is shown at Table 3.

We summarize the table with several key observations.

1) *Single-pass.* By comparing UrbanFM with heuristic and single-pass baselines, it can be seen that UrbanFM consistently outperforms *all* methods in all metrics in all 16 groups of experiments. Take the strongest baseline SRResNet for example. By averaging across all experiments, UrbanFM advances it by 2%, 9%, and 37% on RMSE, MAE, and MAPE respectively. Accordingly, the first row in Table 3 validates the significance of this result. Though the backbone structures are similar, the advances of UrbanFM over SRResNet indeed underline the effectiveness of the proposed distributional upsampling component and the usefulness of the features extracted by the external factor fusion module.
2) *Progressive.* In the category of progressive upsampling, it can be seen that the LapSRN is the stronger baseline, which shows the betterment of using cascading strategy in our task, as it allows a thoroughly trained network. Nevertheless, LapSRN is beaten by UrbanPy in almost all metrics across all experiments. Specifically, the average improvements shown by UrbanPy are 2%, 3%, and 10% on the three metrics. This improvement comes not only from the spirits that are inherited from UrbanFM, but also the unique structure we design for UrbanPy.
3) *Single-pass vs Progressive.* By comparing progressive methods against the single-pass methods, we can see that progressive networks generally demonstrate improvements at larger scales upsampling (typically at 4× and larger). For example, DeepSD versus the SRCNN counterparts and cascading methods versus all the single-pass baselines. In particular, UrbanPy outperform *all* single-pass baselines in this regard. This can be attributed to that progressive method allows the model to conduct upsampling in a smoother way instead of abruptly enlarging the output scale by a large factor. It also worths noting that UrbanFM remains very competitive at 4× upscaling compared to the progressive baselines. This emphasizes that the proposed N^2-*Normalization* and the external factor fusion can provide significant enhancement even without smoothing the upsampling task.

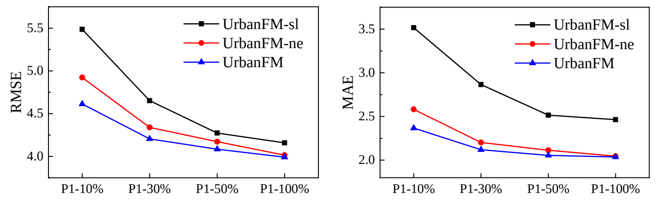
Test of Significance

The Wilcoxon test is an alternative for paired t-test when samples are from a non-normal distribution. Given two methods [A-B], we aim to test the alternative hypothesis: ”The error produced by A is significantly **smaller** than that of B” across the 16 experiments settings (four scales for four periods) and then report the p-values of the null hypothesis at Table 3. The first and second rows are testing using our method and the best baseline in the respective category. The third row compares UrbanPy against UrbanFM. All tests are significant at level $\alpha = 0.05$.



(a) 16-64 setting (b) 16-128 setting

Fig. 7. **Study on Distributional Upsampling.** Performance comparison on whether or not applying the structural constraint.



(a) Results on RMSE (b) Results on MAE

Fig. 8. **Study on External Factor Fusion**. We reveals the contribution of the fusion network by varying the amount of available training data.

5.2.2 Studies on Effectiveness

Study on Distributional Upsampling

To examine the effectiveness of the distributional upsampling module, we compare SRResNet with UrbanFM-ne (using distributional upsampling but no external factors) and UrbanFM-sl (using structural loss instead of distributional upsampling), as shown in Figure 7. In both M-F settings, it can be seen that UrbanFM-sl regularized by L_s performs very close to the SRResNet which is not constrained at all. Though under the setting of 16-64, Urban-sl achieves a smaller error than SRResNet in a subtle way, under the 16-128 setting they behave the opposite. On the contrary, UrbanFM-ne consistently outperforms the others on all three metrics. This results has verified the superiority of the distributional upsampling module over L_s for imposing the structural constraint.

Study on External Factor Fusion

External impacts, though are complicated, can assist the network for better inferences when they are properly modeled and integrated, especially in a more difficult situation when there is less data budget. Thereby, we study the effectiveness of external factors by randomly subsampling from the original training set according to four ratios (i.e., 10%, 30%, 50%, and 100%) which corresponds to four difficulty levels: hard, semi-hard, medium and easy.

As shown in Figure 8, the *gap* between UrbanFM and UrbanFM-ne becomes larger as we reduce the number of training data, indicating that external factor fusion plays a more important role in providing a priori knowledge. When the training size grows, the weight for the prior knowledge decreases, as there exists overlaying information

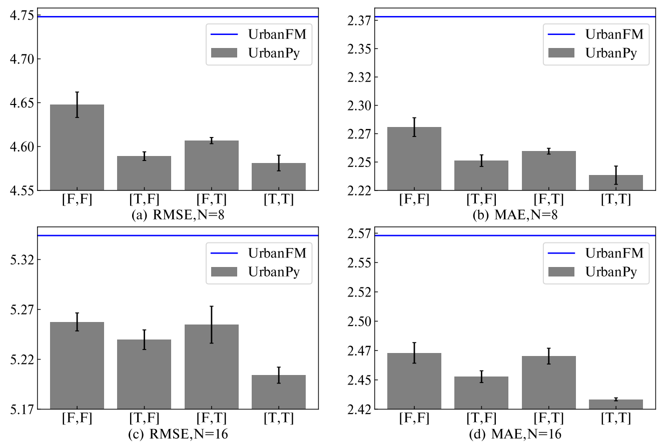


Fig. 9. **Ablation study on UrbanPy**. We conduct three experiments on N=8 and N=16 tasks and report the mean and std of RMSE and MAE on the test sets. The configurations involve [Local Structure, Distributional Loss]. We use *T* to denote the present of an element or *F* otherwise.

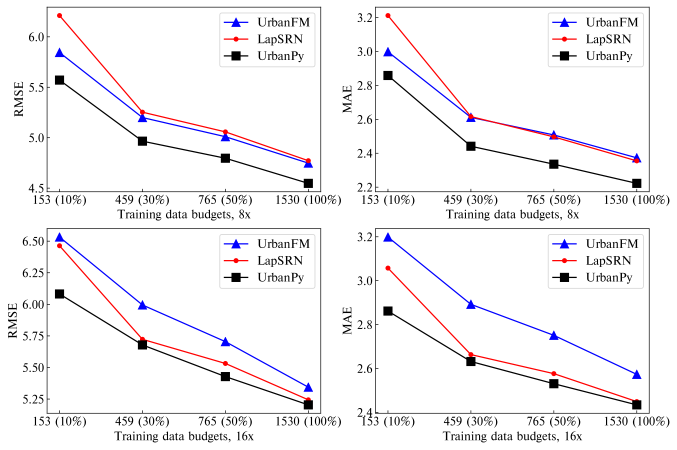


Fig. 10. **Robustness along the change of data budget.** We train the models by feeding different amounts of the training data of P1 for large upscaling tasks including $8\times$ and $16\times$.

between observed traffic flows and external factors. Thus, the network may learn to capture some external impacts when given enough data. Moreover, this trend also occurs between UrbanFM and UrbanFM-sl, which illustrates that the N^2-*Normalization* layer provides a strong structural prior to facilitate network training.

Ablation study on UrbanPy

UrbanPy embraces three enhancements over UrbanFM. To reveal the individual contribution of each element, we conduct ablation studies and diagram the results in Figure 9. It shows that UrbanPy with the pyramid architecture alone (i.e., [F,F]) has outperformed UrbanFM by a large margin in all four settings. This result underscores the importance of the progressive structure and demonstrates its leading role of the improvement of the inference task. By comparing different variants of UrbanPy, we witness that the local structure contributes (i.e., [T,F]) a bit more than the distributional loss (i.e., [F,T]). It can also provides more robustness as is witnessed in Figure 9c. This is not unexpected as the model needs to balance between RMSE and the KL-divergence and thus can variate in some situations. Nevertheless, the combination of both elements (i.e., [T,T]) achieves the best performance and good stableness in all settings.

Methods	Settings	#Params	RMSE	MAE	MAPE
UrbanFM	16-64 (base)	1.6M	4.107	2.118	0.322
SRResNet	20-64	1.8M	4.317	2.586	0.725
UrbanFM	20-64	1.9M	4.094	2.101	0.321
SRResNet	16-128	6.0M	4.301	2.588	0.740
UrbanFM	16-128	6.2M	4.069	2.092	0.316
SRResNet	16-256	24.2M	4.178	2.418	0.614
UrbanFM	16-256	24.4M	4.068	2.087	0.316

(a) Various M-F configurations of UrbanFM at $4\times$

Method	Settings	#Params	RMSE	MAE	MAPE
UrbanPy	4-64-4 (base)	4.7M	4.572	2.237	0.352
	4-32-4	3.0M	4.731	2.326	0.379
	4-128-4	11.6M	4.547	2.222	0.348
	2-64-4	4.3M	4.619	2.265	0.359
	8-64-4	5.6M	4.581	2.240	0.351
	4-64-1	3.9M	4.576	2.239	0.350
	4-64-8	5.8M	4.571	2.233	0.349

(b) Various F-M_2-R configurations of UrbanPy at $8\times$

TABLE 4
Study on Configurations. For both UrbanFM and UrbanPy, we are interested in the key configurations that control the network capability and size of parameters are the filter size F, number of residual blocks M and M_2, and the number of layers R of the proposal network.

Study on Configurations

Table 4a compares the average performance over P1 to P4. Across all hyperparameter settings, UrbanFM consistently outperforms SRResNet, advancing by at least 2.6%, 13.7%, and 48.6%. Besides, this experiment reveals that adding more ResBlocks (larger M) or increasing the number of filters (larger F) can improve the model performance. However, these also increase the training time and memory space. Considering the tradeoff between training cost and performance, we use 16-64 for UrbanFM as default.

In Table 4b, we compare different M_2-F-R combinations with 4-64-4 being the base setting. We observe the following. 1) By increasing F we can see that the network also improves its performance, while the network parameter also blows up quickly. As F=128 only outdoes F=64 marginally, we use the latter setting as default. 2) Different from the effect of F, increasing M_2 can temper the network performance. This is unlikely due to overfitting as the parameter size remains less than that of 4-128-4. Instead, the network can introduce too much zero inputs at the high-level layers as the receptive field at the middle level can already cover the whole coarse-grained input area when M_2 is too large. Therefore, We find 4 a reasonable depth for the residual block. 3) The proposal network is not sensitive to the change of R as is shown in the last two rows. We use R=4 since we find it is more stable for network training.

Study on Robustness to Training Data Budget

With the interest of model performance when less training data are available for larger-scale inference tasks, we depict the comparative results with the strongest baseline LapSRN in Figure 10. As it illustrates, all models increase their performances as the training data become larger. At N=8, UrbanFM remains very competitive and even more data-efficient than the LapSRN that enjoys the progressive structure. At N=16, LapSRN achieves lower error than UrbanFM as the structure advantages start to overcomes the benefits bought by N^2-Normalization and external features when the task becomes too difficult. Therefore, it is no surprise that UrbanPy outperforms the LapSRN in all data budgets for both metrics, as it combines the advances of UrbanFm and benefits from progressive upsampling.

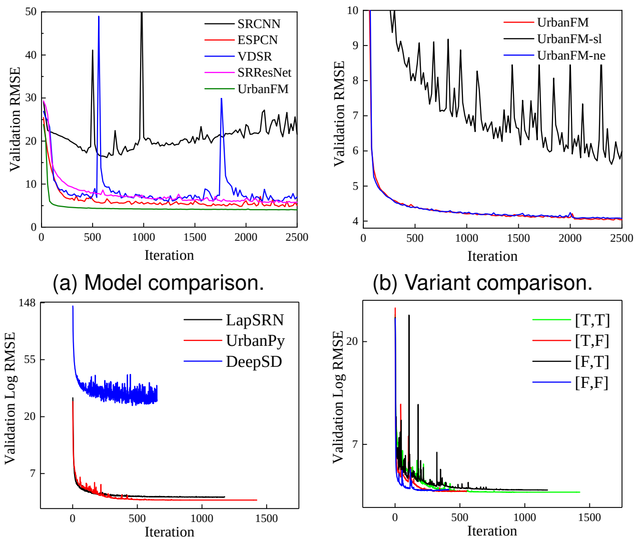


Fig. 11. **Convergence speed of various methods.** Figure (a) and (b) show the convergence speed for single-pass methods using 4× upscaling factors. Differently, we plot logarithm scores in (c) and (d) for progressive methods with 8× for a clearer illustration. Note that we double the training batch size for progressive methods and employ early stopping according to the best validation scores, as their training are generally slower. We use the same notation for variants as in Figure 9.

5.2.3 Study on Training Efficiency

Figure 11 plots the RMSE on the validation set during the training phase using P1-100%. Figure 11(a) and 11(b) delineate that UrbanFM converges much *smoother* and *faster* than the single-pass baselines and its variants. Specifically, 11(b) suggests such efficiency improvement can be mainly attributed to the N^2-*Normalization* layer since UrbanFM-sl converges much slower and fluctuates drastically even it is constrained by L_s, when compared with UrbanFM and UrbanFM-ne. This also suggests that learning the spatial correlation is a non-trivial task. Moreover, UrbanFM-ne behaves closely to UrbanFM as external factors fusion affects the training speed subtly when training data are abundant as suggested by the previous experiments.

The convergence curves for progressive methods are depicted by Figure 11(c). It illustrates that DeepSD can not converge to the same level as the cascading methods do, due to the inconsistency resides between the stacked components while cascading structure allows training a more coherent network. UrbanPy converges as fast as LapSRN and can be trained continuously longer, as the proposal network is a more powerful component than the simple bilinear interpolation function used by LapSRN. 11(d) gives a more detailed plot that focuses on UrbanPy and its variants. It can be seen that the [F,F] setting converges smoother then others, however, stops earlier and fails to improve further. [F,T] shows large fluctuation during training as the KL-divergence can reduce the stableness. Nevertheless, [T,T] shows both smooth convergence curve and continuous improvement, which explains why combining both local structure and distributional loss can outperform the state-of-the-art methods.

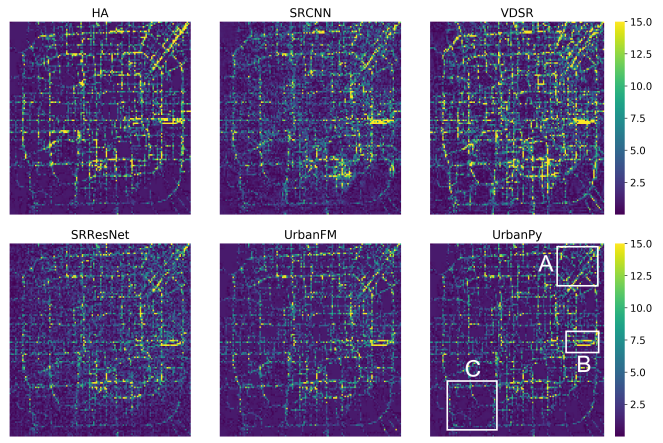


Fig. 12. Visualization for inference errors among different methods. Best view in color.

5.2.4 Visualization

1) *Inference error.* Figure 12 displays the inference error $\|\mathbf{X}^f - \tilde{\mathbf{X}}^f\|_{1,1}$ from our methods and the other four baselines for a sample at the 4× task, where a brighter pixel indicates a larger error. Contrast to the baseline methods, both UrbanFM and UrbanPy achieves higher fidelity for totality and in detail, which corresponds to the quantitive results from Table 2. For instance, areas A and B are ”hard areas” to be inferred, as A (Sanyuan bridge, the main entrance to downtown) and B (Sihui bridge, a huge flyover) are two of the top congestion points in Beijing. Traffic flow of these locations usually fluctuates drastically and quickly, resulting in higher inference errors. Nonetheless, our methods remain to produce better performances in these areas. Another observation is that the SR methods (SRCNN, VDSR, and SRResNet) tend to generate blurry images as compared to structural methods (HA and our methods). For instance, even if there is zero flow in area C, SR methods still generate error pixels as they overlap the predicted patches. This suggests the FUFI problem does differ from the ordinary SR problem and requires specific designs.

2) *External influence.* Figure 13(a)-(d) portray that the *inferred* distribution over subregions varies along with external factor changes. To stay succinct, we present the results of UrbanFM only as UrbanPy produces similar visualization regarding external factors. On weekdays, at 10 a.m., people had already flowed to the office area to start their work (b); at 9 p.m., many people had returned home after a hard-working day (c). On weekends, most people stayed home at 10 a.m. but some industrial researchers remained working in the university labs. This result proves that our methods indeed capture the external influence and learns to adjust the inference accordingly.

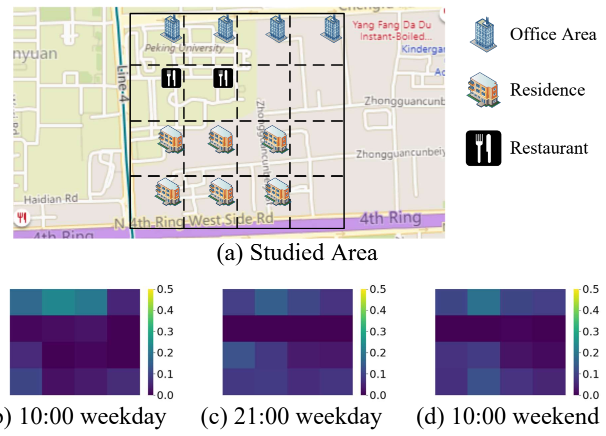


Fig. 13. Case study on a superregion near Peking University. See our github for further dynamic analysis on this area.

TABLE 5
Results comparison on Happy Valley. We evaluate the task with 2× upscaling for this area. All models are selected based on the best validation performance and test results are presented.

Methods	Settings	Params	RMSE	MAE	MRE
MEAN	x	x	9.206	2.269	0.799
HA	x	x	8.379	1.811	0.549
SRCNN	768	7.4M	8.291	2.175	0.816
ESPCN	768	7.4M	8.156	2.155	0.805
VDSR	16-64	0.6M	8.490	2.128	0.756
SRResNet	16-128	5.5M	8.318	1.941	0.679
UrbanFM-sl	16-128	5.5M	8.312	1.939	0.677
UrbanFM-ne	16-128	5.5M	8.138	1.816	0.537
UrbanFM	16-128	5.6M	**8.030**	**1.790**	**0.531**
LapSRN	10-128	3.2M	8.249	1.832	0.547
UrbanPy-FF	8-64-4	1.2M	8.280	1.879	0.587
UrbanPy-TT	8-64-4	1.3M	**8.028**	1.749	0.523
UrbanPy-FF	8-128-4	4.4M	8.184	1.900	0.618
UrbanPy-TT	8-128-4	4.4M	8.332	**1.732**	**0.508**

5.2.5 *Results on HappyValley*

Table 5 shows model performances using the HappyValley dataset. Note that in this experiment, we do not include DeepSD, since this task contains only 2× upscaling such that DeepSD degrades to SRCNN in this case. One important trait of the HappyValley dataset is that it contains more spikes on the fine-grained flow distribution, which results in a much larger RMSE score versus that in the TaxiBJ task. Nonetheless, in the *single-pass* branch UrbanFM remains the winner method outperforming the best baseline by 3.5%, 7.8%, and 22%; the UrbanFM-ne still holds the runner-up position. Moreover, it is not unexpected to see LapSRN is worse than UrbanFM as the former shows no progressive superiority over UrbanFM in this task. Move on to the *progressive* branch. Though the [F,F] variants show worse performances than UrbanFM, as the compositional architecture can complicate the training when the task is as simple as 2× upscaling, the full models of UrbanPy can provide better scores than its single-pass counterpart, which validates the usefulness of the two components even when no structural advance can be exploited.

To summarize, this collection of experiment prove that our methods not only work on the large-scale scenario, but is also adaptable to smaller areas, which concludes our empirical studies.

Limitation
While our methods demonstrate leading performance for both low-scale (UrbanFM) and large-scale (UrbanPy) urban flow inference tasks, the current structure accepts the regular partition of the urban area. For non-regular partition, we need to use a graph to represent the locations as nodes and connections between locations (e.g., road networks) as the edges. Besides, the UrbanPy learns slower than its single-pass counterpart (typically 4 times slower as can be seen in Figure 11) as the dynamics become more complicated with the pyramid structure, which is also noted by [17, 18]. Nevertheless, this is a trade-off between training efficiency and inference performance. We suggest that when the required upsampling scale is large, the UrbanPy is a more favorable choice; if the training time is of the key concern or the scale is small, we should opt for the UrbanFM model.

6 Related Work

6.1 Urban Flows Analysis

6.1.1 *Urban Flows Applications*

Development of modern ubiquitous technologies, such as wireless sensing networks, internet of things, and crowdsourcing, have provided various ways for people to acquire an image of urban dynamics. With rich data becoming available and increasing demand for understanding and exploiting such dynamics, urban flow analysis has recently attracted the attention of a large amount of researchers [6]. Zheng et al. [6] first transformed public traffic trajectories into other data formats, such as graphs and tensors, to which more data mining and machine learning techniques can be applied. Based on our observation, there were several previous works [28, 29] forecasting billions of individual mobility traces rather than aggregated flows in a region.

Recently, researchers have started to focus on city-scale traffic flow prediction [30]. Inspired by deep learning techniques that power many applications in modern society [31], a novel deep neural network was developed by Zhang et al. [32] to simultaneously model spatial dependencies (both near and distant), and temporal dynamics of various scales (i.e., closeness, period and trend) for citywide crowd flow prediction. Following this work, Zhang et al. [8] further proposed a deep spatio-temporal residual network to collectively predict inflow and outflow of crowds in every city grid. Apart from the above applications, very recently Liang et al. [1] presented UrbanFM, the first work to the best of our knowledge to solve the novel FUFI problem in urban scenario. In this paper we further extend the capability of UrbanFM to solve larger-scale inference tasks by presenting the UrbanPy framework.

6.1.2 *Urban Flows Reconstruction*

Urban flow data reconstruction is particularly useful when only coarse-grained data is available. For this aim, previous studies mainly target at reconstructing high-resolution population density, and can be categorized into two strands. In the first strand, researchers relied on various auxiliary data, such as census statistics and site reports, to heuristically reconstruct high-resolution human flows [33]. However, such method requires expensive human labor and usually extremely time-consuming. More recently, researchers introduce the power of machine learning to infer fine-grained data. As a representative, Stevens et al. used random forest algorithms as the inference model to disaggregate census data [34]. The state-of-the-art, namely DeepDPM [26], borrowed the idea from image super-resolution and stacked multiple SRCNNs to progressively upsample the coarse-grained input to the desired granularity. Compared to Zong et al. [26], we not only identify and exploit the critical structure of the problem, but also incorporate various external factors to further boost the inference performance.

6.2 Image Super-Resolution

Inspiring the work of DeepDPM [26], single image super-resolution (SISR) [35, 36] does share a similar nature of the FUFI problem we study in this paper, as both are targeting to recover high-resolution (HR) raster-based data from its low-resolution (LR) counterparts. To provide a broader view in this correlated domain, we discuss image super-resolution techniques from two categories: single-pass and progressive methods. Noted that though the two problems have correlations, FUFI differs from SISR that it exhibits two aforementioned challenges: structural constraints and external factors. Therefore, a simple application of related methods to FUFI is infeasible, and thus a careful redesign of the model architecture is needed.

6.2.1 Single-pass methods

Single-pass methods process coarse-grained images in one or multiple consecutive upsampling steps. Early upsampling techniques exploited interpolation methods such as bicubic interpolation and Lanczos resampling [37]. Also, several studies utilized statistical image priors [38, 39] to achieve better performances. Advanced works aimed at learning the non-linear mapping between LR and HR images with neighbor embedding [40] and sparse coding [41, 42]. However, these approaches are still inadequate to reconstruct realistic and fine-grained textures of images.

Recently, a series of models based on deep learning has achieved great success in terms of SISR as they do not require any human-engineered features and show the state-of-the-art performance. Since Dong et al. [23] first proposed an end-to-end mapping method represented as CNNs between the low-resolution (LR) and high-resolution (HR) images, various CNN based architectures have been studied for SR. Among them, Shi et al. [14] introduced an efficient sub-pixel convolutional layer which is capable of recovering HR images with very little additional computational cost compared with the deconvolutional layer at training phase. Inspired by VGG-net for ImageNet classification [43], a very deep CNN was applied for SISR in [24]. However, training a very deep network for SR is really hard due to the small convergence rate. Kim et al. [24] showed residual learning speed up their training phase and verified that increasing the network depth could contribute to a significant improvement in SR accuracy.

The general process of SISR methods (i.e., feature extraction followed by SR image recovery) inspires our solution for FUFI. However, these approaches are not suitable for the FUFI problem since the flow data present a very specific hierarchical structure with regard to natural images, as such, the related arts cannot be simply applied to our application in terms of efficiency and effectiveness.

6.2.2 Progressive methods

Though single-pass methods demonstrate useful performances at small-scale upsampling (typically $2\times$ and $4\times$), these methods encounter difficulties when dealing with large-scale super-resolution tasks (e.g., $8\times$) [7]. This can be attributed to the abrupt upsampling based on low-level features and utilize only one supervision signal at the output end. To tackle this problem, several nascent works [7, 17, 18] proposed progressive models based on laplacian pyramid, where the network aimed to learn the upsampled residuals and perform upsampling by aggregating the residuals with interpolated images. This inspires the cascading design of our UrbanPy architecture.

Apart from super-resolving classical images, there are limited studies that utilized super-resolution methods to solve real-world problems in the urban area. In particular, two very recent works [25, 26] employ the same strategy of stacking SRCNN [23] for two different tasks: Vandal et al. [25] aimed at statistical downscaling of climate and earth system simulations based on observational and topographical data; likewise, Zong et al. [26] addressed the task of inferring fine-grained population density by treating the population heat maps as images.

Different from the related arts that directly target the modeling on pixel values, we instead model the distributions over the superregions and their fine-grained counterparts, by doing which we are able to capture the essence of the FUFI problem. Moreover, we also include the external features which are very unique in the urban scenario.

7 Conclusion

Fine-grained urban flow information benefits urban development, while one may need to trade off between data utility and storage efficiency. To bridge such gap, in this paper, we have formalized the fine-grained urban flow inference problem and two versions of deep neural network-based methods to solve it. The preliminary version (i.e., UrbanFM) focuses on addressing the two specific challenges of the problem through embedding the hierarchical structure in the model and generating a comprehensive representation for external factors. Build upon the key components of UrbanFM, we present a more advanced version named UrbanPy by employing the progressive upsampling strategy, which resolves the defects of UrbanFM when tackling larger-scale inference tasks. We have conducted extensive experiments, both qualitatively and quantitively to study the actual performance of the models using the TaxiBJ dataset and HappyValley datasets. The empirical studies and visualizations have supported the advantages of both UrbanFM and UrbanPy on both efficiency and effectiveness. Codes are also published for the community [6].

We have also discussed the limitation of the current work, which is mainly due to the learning dynamic of the pyramid structure and remains an open problem. For our

6. https://github.com/ouyangksoc/UrbanPy

future work, we are interested in improving the learning efficiency of the UrbanPy framework, e.g., by curriculum strategy [18]. Also in this work we focus on pushing the boundary of inference using only a single coarse-grained flow map, we will also like to explore the temporal relationship of the coarse-grained data to try to further improve the inference performance.

REFERENCES

[1] Y. Liang, K. Ouyang, L. Jing, S. Ruan, Y. Liu, J. Zhang, D. S. Rosenblum, and Y. Zheng, "UrbanFM: Inferring fine-grained urban flows," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019, pp. 3132–3142.

[2] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz, "Hadoop-gis: A high performance spatial data warehousing system over mapreduce," in *Proceedings of the VLDB Endowment International Conference on Very Large Data Bases*, vol. 6, no. 11.

[3] S. You, J. Zhang, and L. Gruenwald, "Large-scale spatial join query processing in cloud," in *2015 31st IEEE International Conference on Data Engineering Workshops*, 2015.

[4] R. Li, H. He, R. Wang, Y. Huang, J. Liu, S. Ruan, T. He, J. Bao, and Y. Zheng, "Just: Jd urban spatio-temporal data engine," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020.

[5] W. Pedrycz and S.-M. Chen, *Information granularity, big data, and computational intelligence*. Springer, 2014, vol. 8.

[6] Y. Zheng, L. Capra, O. Wolfson, and H. Yang, "Urban computing: concepts, methodologies, and applications," *ACM Transactions on Intelligent Systems and Technology*, vol. 5, no. 3, p. 38, 2014.

[7] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Deep laplacian pyramid networks for fast and accurate superresolution," in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, no. 3, 2017, p. 5.

[8] J. Zhang, Y. Zheng, and D. Qi, "Deep spatio-temporal residual networks for citywide crowd flows prediction," in *The AAAI Conference on Artificial Intelligence*, 2017, pp. 1655–1661.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[10] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. P. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network." in *CVPR*, vol. 2, no. 3, 2017, p. 4.

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[12] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, p. 947, 2000.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *ECCV*, 2016, pp. 630–645.

[14] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *ICCV*, 2016, pp. 1874–1883.

[15] "Nearest-neighbor interpolation," Nov 2019. [Online]. Available: https://en.wikipedia.org/wiki/Nearest-neighbor_interpolation

[16] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "Geoman: Multi-level attention networks for geo-sensory time series prediction," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 2018.

[17] W.-S. Lai, J.-B. Huang, N. Ahuja, and M.-H. Yang, "Fast and accurate image super-resolution with deep laplacian pyramid networks," *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[18] Y. Wang, F. Perazzi, B. McWilliams, A. Sorkine-Hornung, O. Sorkine-Hornung, and C. Schroers, "A fully progressive approach to single-image super-resolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 864–873.

[19] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of machine learning research*, vol. 11, no. Dec, pp. 3371–3408, 2010.

[20] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[21] C. M. Bishop, *Pattern recognition and machine learning*. Springer Science+ Business Media, 2006.

[22] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich, "Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks," in *International Conference on Machine Learning*, 2018, pp. 793–802.

[23] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 295–307, 2016.

[24] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *CVPR*, 2016, pp. 1646–1654.

[25] T. Vandal, E. Kodra, S. Ganguly, A. Michaelis, R. Nemani, and A. R. Ganguly, "Deepsd: Generating high resolution climate change projections through single image super-resolution," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 1663–1672.

[26] Z. Zong, J. Feng, K. Liu, H. Shi, and Y. Li, "Deepdpm: Dynamic population mapping via deep neural network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 1294–1301.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[28] X. Song, Q. Zhang, Y. Sekimoto, and R. Shibasaki, "Prediction of human emergency behavior and their mobility following large-scale disaster," in *Proceedings of the ACM SIGKDD*, 2014.

[29] Z. Fan, X. Song, R. Shibasaki, and R. Adachi, "Citymo-

mentum: an online approach for crowd behavior prediction at a citywide level," in *Proceedings of the ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015, pp. 559–569.

[30] M. X. Hoang, Y. Zheng, and A. K. Singh, "Fccf: forecasting citywide crowd flows based on big data," in *Proceedings of the ACM SIGSPATIAL*, 2016.

[31] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[32] J. Zhang, Y. Zheng, D. Qi, R. Li, and X. Yi, "Dnn-based prediction model for spatio-temporal data," in *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2016, p. 92.

[33] Y. Fang and J. W. Jawitz, "High-resolution reconstruction of the united states human population distribution, 1790 to 2010," *Scientific data*, vol. 5, no. 1, pp. 1–15, 2018.

[34] F. R. Stevens, A. E. Gaughan, C. Linard, and A. J. Tatem, "Disaggregating census data for population mapping using random forests with remotely-sensed and ancillary data," *PLOS one*, vol. 10, no. 2, 2015.

[35] B. K. Gunturk, A. U. Batur, Y. Altunbasak, M. H. Hayes, and R. M. Mersereau, "Eigenface-domain super-resolution for face recognition," *IEEE Transactions on Image Processing*, vol. 12, no. 5, pp. 597–606, 2003.

[36] M. W. Thornton, P. M. Atkinson, and D. Holland, "Sub-pixel mapping of rural land cover objects from fine spatial resolution satellite sensor imagery using super-resolution pixel-swapping," *International Journal of Remote Sensing*, vol. 27, no. 3, pp. 473–491, 2006.

[37] C. E. Duchon, "Lanczos filtering in one and two dimensions," *Journal of Applied Meteorology*, vol. 18, no. 8, pp. 1016–1022, 1979.

[38] J. Sun, Z. Xu, and H.-Y. Shum, "Image super-resolution using gradient profile prior," in *CVPR*, 2008, pp. 1–8.

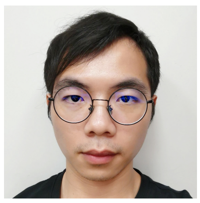
[39] Y.-W. Tai, S. Liu, M. S. Brown, and S. Lin, "Super resolution using edge prior and single image detail synthesis," in *CVPR*, 2010, pp. 2400–2407.

[40] H. Chang, D.-Y. Yeung, and Y. Xiong, "Super-resolution through neighbor embedding," in *CVPR*, 2004.

[41] J. Yang, J. Wright, T. S. Huang, and Y. Ma, "Image super-resolution via sparse representation," *IEEE Transactions on Image Processing*, vol. 19, no. 11, pp. 2861–2873, 2010.

[42] R. Timofte, V. De Smet, and L. Van Gool, "A+: Adjusted anchored neighborhood regression for fast super-resolution," in *Asian Conference on Computer Vision*, 2014, pp. 111–126.

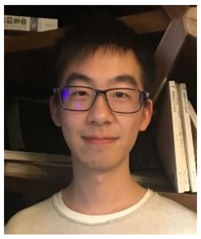
[43] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.



Kun Ouyang obtained his PhD degree in Computer Science at the National University of Singapore (NUS). He was also a research scholar in SAP Singapore. He obtained his B.E. degree in the Internet of Things, in Wuhan University. His research interests include human mobility analytics, spatial-temporal data mining and deep learning. Now he is dedicated to work in online advertising system in E-commerce.

Yuxuan Liang is currently pursuing his Ph. D. degree at School of Computing, National University of Singapore. His research interests mainly lie in machine learning, deep learning and their applications in urban areas.

Zekun Tong received the B.E. degree in computer science and technology from Xidian University, Xi an, China, in 2018. He is currently pursuing the Ph.D. degree in Industrial System Engineering with the School of Engineering, National University of Singapore, Singapore. His research interests include machine learning, optimization and data analytics.



Sijie Ruan is a Ph.D. student in the School of Computer Science and Technology, Xidian University. He received his B.E. degree from Xidian University in 2017. His research interests include urban computing, spatio-temporal data mining, and distributed systems. He was an intern in MSR Asia from 2016 to 2017. He is now a research intern in JD Intelligent Cities Research, under the supervision of Prof. Yu Zheng and Dr. Jie Bao.

Ye Liu Dr. Ye Liu received his Ph.D degree from National University of Singapore. Before that he received the M.Sc. degree from Peking University. His research interests lie mainly in the areas of urban computing, ubiquitous computing, knowledge graph, data mining, machine learning, artificial intelligence and their applications in human activity analysis, social network analysis, knowledge graph, urban computing, computational advertising, recommendation.

Yu Zheng is a Vice President of JD.COM and Chief Data Scientist at JD Digits, passionate about using big data and AI technology to tackle urban challenges. His research interests include big data analytic, spatio-temporal data mining, machine learning, and artificial intelligence. He also leads the JD Urban Computing Business Unit as the president and serves as the director of the JD Intelligent City Research. Before joining JD, he was a senior research manager at Microsoft Research. Zheng is also a Chair Professor at Shanghai Jiao Tong University, an Adjunct Professor at Hong Kong University of Science and Technology.

David S. Rosenblum is Provost's Chair Professor of Computer Science at the National University of Singapore (NUS). His research interests span many problems in software engineering, distributed systems and ubiquitous computing, and his current research focuses on probabilistic verification, uncertainty in software testing, and machine learning. He is a Fellow of the ACM and IEEE and was previously Editor-in-Chief of the ACM Transactions on Software Engineering and Methodology (ACM TOSEM) and Chair of the ACM Special Interest Group in Software Engineering (ACM SIGSOFT). He has received two "test-of-time" awards for his research papers, including the ICSE 2002 Most Influential Paper Award for his ICSE 1992 paper on assertion checking, and the inaugural ACM SIGSOFT Impact Paper Award in 2008 for his ESEC/FSE 1997 on Internet-scale event observation and notification (co-authored with Alexander L. Wolf). He also received the ACM SIGSOFT Distinguished Service Award in 2018.