DeepGPS: Deep Learning Enhanced GPS Positioning in Urban Canyons

Zhidan Liu, *Member, IEEE*, Jiancong Liu, Xiaowen Xu, and Kaishun Wu, *Member, IEEE*

Abstract—Global Positioning System (GPS) has benefited many novel applications, e.g., navigation, ride-sharing, and location-based services, in our daily life. Although GPS works well in most places, its performance in urban canyons is well-known poor, due to the signal reflections of non-line-of-sight (NLOS) satellites. Tremendous efforts have been made to mitigate the impacts of NLOS signals, while previous works heavily rely on precise proprietary 3D city models or other third-party resources, which are not easily accessible. In this paper, we present *DeepGPS*, a deep learning enhanced GPS positioning system that can correct GPS estimations by only considering some simple contextual information. *DeepGPS* fuses environmental factors, including building heights and road distribution around GPS's initial position, and satellite statuses to describe the positioning context, and exploits an encoder-decoder network model to implicitly learn the complex relationships between positioning contexts and GPS estimations from massive labeled GPS samples. As a result, the well-trained model can accurately predict the correct position for each erroneous GPS estimation given its positioning context. We further improve the model with a novel constraint mask to filter out invalid candidate locations, and enable continuous localization with a simple mobility model. A prototype system is implemented and experimentally evaluated using a large-scale bus trajectory dataset and real-field GPS measurements. Experimental results demonstrate that *DeepGPS* significantly enhances GPS performance in urban canyons, e.g., on average effectively correcting 90.1% GPS estimations with accuracy improvement by 64.6%.

Index Terms—GPS, positioning, deep learning, urban canyons, NLOS satellite

1 Introduction

Global Navigation Satellite Systems (GNSSs), such as the well-known Global Positioning System (GPS), have benefited many intelligent applications, including navigation [65], instant delivery [69], ride-sharing [40], autonomous driving [12], and location-based services [34]. Accurate positioning is of great importance for those applications to provide effective and efficient services. While GPS generally works well in most places, it is still extremely unreliable in urban canyons, e.g., GPS errors in such challenging environments can be larger than 50 meters [28].

GPS error in urban canyons primarily comes from the effects of notorious multipath interference or non-line-of-sight (NLOS) receptions, due to satellite signal reflections by high-rising buildings [33]. Although multipath interference can be well addressed by some hardware or software designs [18], [27], [44], the NLOS satellite signals at GPS receivers remain the major source of positioning errors in urban canyons. In principle, a GPS receiver needs to receive signals from at least four satellites for accurately triangulating its location [33]. In urban canyon scenarios, the signal of an NLOS satellite may be reflected by one or even more tall buildings, and causes the so-called *pseudorange*, which is an approximation of the distance between the target satellite and GPS receiver, to be larger, resulting in positioning errors.

In the literature, tremendous efforts have already been devoted to mitigate the impact of NLOS receptions. Early works [21] apply ray-tracing algorithms on satellite signals to compute pseudorange errors, which have to reconstruct the reflective surfaces of street buildings using specialized hardware, e.g., panoramic cameras [54] and LiDAR [57]. Based on the proprietary 3D city models, most of recent works exploit building geometry to calculate the similarity of signal paths [29], [39], [45], [48] or satellite visibility [23], [49], [60], [63], [68] between candidate locations around GPS's initial position and the receiver's observations, and output the spot with the best match as the solution. These works, however, heavily rely on the precise proprietary 3D city models or other third-party resources, e.g., panoramic images [39], which are not easily accessible and thus largely limit their practical adoptions.

Despite the limitations of previous works, they motivate us to imagine that there should exist some mapping function f that can map a GPS estimation and its surrounding environment to the ground truth position where the receiver actually stands. It is possible and reasonable, since the majority of key environmental factors, e.g., buildings, are relatively stationary and satellites are regularly operating in

- *Zhidan Liu, Jiancong Liu, and Xiaowen Xu are with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, Guangdong 518060, China. E-mail: liuzhidan@szu.edu.cn, {liujiancong2018, xuxiaowen2019}@email.szu.edu.cn.*
- *Kaishun Wu is with The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, and also with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, Guangdong 518060, China. E-mail: wu@szu.edu.cn.*

Manuscript received 26 March 2022; revised 13 July 2022; accepted 14 September 2022. Date of publication 21 September 2022; date of current version 5 December 2023.
This work was supported in part by China NSFC under Grants 62172284, 61872248, and U2001207, in part by the Grant of Guangdong Basic and Applied Basic Research Foundation under Grants 2022A1515010155 and 2017A030312008, in part by Shenzhen Science and Technology Foundation under Grants ZDSYS20190902092853047 and R2020A045, in part by the Project of DEGP under Grants 2019KCXTD005 and 2021ZDZX1068, and in part Guangdong "Pearl River Talent Recruitment Program" under Grant 2019ZT08X603.
(Corresponding author: Kaishun Wu.)
Digital Object Identifier no. 10.1109/TMC.2022.3208240

most of the time. Therefore, given a location and its contextual information, e.g., heights of buildings and satellite statuses, the ground truth position can be inferred. Of course, it is impossible to explicitly enumerate all positioning contexts to find the mapping function f, due to huge computation and storage costs, while we could build a deep neural network to approximate the mapping function f for enhancing GPS positioning in urban canyons, by leveraging the powerful representation ability of deep learning [35]. Thanks to the widely available GPS trajectories [70], they provide us rich training data to learn the deep neural network model for a given region. The well trained model then can provide swift and accurate GPS positioning for users of that region.

In this paper, we present a deep learning enhanced GPS positioning system - *DeepGPS*, which can achieve accurate positioning in urban canyons. We build a deep neural network to implicitly capture the complex relationships between positioning contexts and the ground truth positions from massive labeled GPS samples. The trained model, serving as a black box, can effectively transform an erroneous GPS estimation to its correct position and thus greatly improve positioning accuracy in urban canyons.

Instantiating *DeepGPS*, however, requires to address three key challenges. First, there are multiple factors, e.g., layouts and heights of buildings, satellite distribution, human dynamics, and etc, which would affect GPS accuracy in urban canyons, while these factors are in different forms of modality, resulting in their representations with varied dimensions. How to properly represent and fuse relevant positioning contexts as the input of deep neural network, yet without information loss, is challenging. To attack this issue, we transform all contextual factors into matrices of the same size, and combine them as a holistic multi-source data input. Such representations allow us to leverage the recent advances on 2D image based techniques in computer vision [32], [36], which are useful on capturing spatial correlations among environmental factors around GPS's initial position.

Second, predicting the correct positions from GPS estimations and other inputs can be modeled as a regression problem. Since position is usually denoted as a pair of latitude and longitude, it is thus difficult to train the regression model, due to the huge search space for float numbers. As a result, how to model the position correcting problem is crucial, since it also determines the architecture of deep neural network. To address this challenge, we analyze GPS error distribution in urban canyons, and generate a set of candidate cells around GPS's initial position. Then, we transform the position correcting problem into a classification problem, where we aim to predict which cell is the most likely to contain the ground truth position. To solve this classification problem, we adopt an encoder-decoder network. The model consists of an encoder, which extracts features from the input matrices, and two decoders, i.e., distance decoder and position decoder, which feed into feature map generated by encoder to predict positioning error and correct position, respectively. More importantly, these two decoders share information from the same encoder, and mutually constrain each other to predict the most possible error and position, which can best match with the observation. Furthermore, we embed the environmental context into a constraint mask that can improve the model's accuracy by filtering out inaccessible cells.

Third, most of location-based applications generally need to track objects of interest, which calls for continuous and accurate positioning. Although we could exploit the model to separately correct each GPS estimation, such a method omits temporal correlations among user's movements and thus is inefficient. Indeed, we can adopt advanced techniques like particle filtering [59] to track user's location, however, these solutions require extra devices to collect mobility data. Thus, it remains a challenge to make the model well support continuous localization. To that end, we propose a simple mobility model based continuous localization method. We use the latest fixed positions to update user's mobility model, which helps to compute a reachable area of the user given time interval between two positioning instances. The reachable area offers different confidences on candidate cells, and helps us to accurately and efficiently determine the cell with correct position.

In summary, this paper makes the following contributions:

- To the best of our knowledge, we are the first to enhance GPS performance in urban canyons by implicitly learning the relationships between diverse positioning contexts and GPS estimations.
- We design and implement *DeepGPS*, which is built on the encoder-decoder network architecture and can accurately predict positioning errors and correct positions from erroneous GPS estimations and multi-source data fusion.
- We incorporate domain knowledge into a novel constraint mask that further improves the neural network, and propose a mobility model to enable continuous localization.
- A prototype system[1] is developed and experimentally evaluated. Extensive evaluation results based on both a large bus GPS trajectory dataset and real-field GPS measurements show the effectiveness of *DeepGPS*. On average, our system can effectively correct 90.1% GPS estimations with positioning accuracy improvement by 64.6%.

The rest of this paper is organized as follows. We present the background and motivation in Section 2. The design of *DeepGPS*is elaborated and implemented in Sections 3 and 4, respectively. The performance evaluations are conducted in Section 5. We review related works in Section 6, and present the discussions in Section 7. Finally, Section 8 concludes this paper.

2 Background and Motivation

In this section, we will introduce the background of GPS, discuss how GPS performs in urban canyons, and then motivate our design of *DeepGPS*by analyzing previous works.

2.1 The Prime of GPS

The GPS navigation system is constituted of three components: satellite constellation, ground stations, and user's GPS receiver. Specifically, satellite constellation consists of 32 satellites, each of which orbits the Earth every 12 hours and continuously broadcasts its position and other metadata above

1. The source code is shared for the benefits of the community [8].

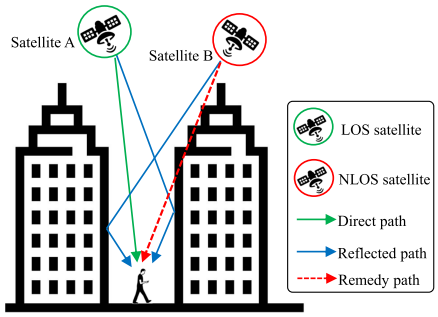

Fig. 1. The multipath and NLOS receptions of satellite signals in urban canyons.

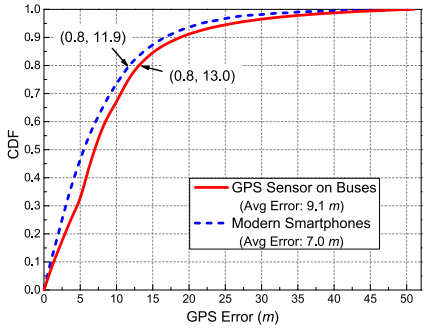

Fig. 2. GPS error statistics in two datasets collected by buses and smartphones.

the Earth [33]. The metadata include various attributes of each satellite, e.g., satellite's position. The ground stations monitor and manage all satellites by sending them with parameters that control their orbits and trajectories. In addition, satellites carry stable atomic clocks that are precisely synchronized with each other and clocks of ground stations.

The GPS receiver receives signals from available satellites, and calculates each signal's travel distance, i.e., *pseudorange*, by multiplying speed of light with the signal's propagation delay. With the positions of observed satellites and their pseudoranges, the receiver estimates its location (x, y, z) using the least square method under constraint conditions as follows:

$$\sqrt{(x-x_i)^2+(y-y_i)^2+(z-z_i)^2}=\ell_i, \tag{1}$$

for $i=1,2,\ldots,m$, where m is the number of visible satellites for the receiver, (x_i, y_i, z_i) is the position of the i-th satellite, and ℓ_i is the measured distance between the receiver and the i-th satellite. In practice, the receiver's clock is not synchronized with satellites' clocks, and thus a receiver must have at least four satellites in view to determine its location and the unknown time deviation between the receiver and the satellites.

A standalone GPS receiver suffers from a lot of error sources that severely affect GPS positioning performance [33]. For example, the inconsistencies of atmospheric conditions will affect the travel speed of GPS signals, resulting in ionospheric delay and tropospheric delay. In addition, the Doppler effect due to the Earth's rotation and the satellite's velocity will also affect the travel time of GPS signals, and introduces positioning error. Moreover, multipath interference, where the same signal is received directly from the satellite and via reflection, and NLOS satellites, whose signals are received via reflection only, will both affect the pseudorange estimation. Lastly, it is worthy to note that although satellite clocks are highly accurate, they are still not perfect with clock error about 8.64 to $17.28\,ns$ per day [3]. For simplicity, we only mention several major error sources in GPS positioning, and more analysis about GPS errors can be found in [3], [33].

There exist various techniques, e.g., differential GPS [25], to compensate for the errors caused by inconsistencies of atmospheric conditions and Doppler effect in modern GPS receivers [39]. Different from GPS receiver clock error that can be fixed by receiving more satellite signals, the satellite clock error can be corrected using a fitted polynomial model [52]. The other two error sources, i.e., multipath interference and NLOS satellites, remain the major challenge for GPS positioning, and are particularly severe in urban canyons [16], [20], [23].

2.2 How GPS Performs in Urban Canyons

In urban canyons, satellite signals are frequently reflected or blocked by the densely distributed buildings. As a result, the GPS receivers cannot receive sufficient readings or mistakenly estimate pseudoranges, leading to larger positioning errors, e.g., more than 50 meters [28]. As shown in Fig. 1, the user's receiver receives a direct signal (i.e., the green line) and a reflected signal (i.e., the blue line) from satellite A. These two signals form multipath receptions for the receiver and interfere its positioning. As satellite B is blocked by buildings, the receiver can only receive reflected signals from satellite B, which is thus called as *NLOS satellite* for the receiver. Compared to the direct signal, a reflected signal will travel more distance, e.g., 100 meters [39], resulting in a much larger estimated pseudorange. Unfortunately, the GPS receivers cannot reliably distinguish between direct and reflected signals. Therefore, both multipath and NLOS receptions become the primary causes of GPS errors in urban areas.

To understand the practical performance of GPS, we analyze two real-world GPS datasets, which are collected from regularly operated buses equipped with GPS sensors and several modern smartphones, respectively. More details about our GPS datasets are described in Section 5.1. The positioning error statistics of these GPS data are plotted in Fig. 2. Since public buses travel along scheduled routes within the downtown areas, we find that the GPS positioning errors are relatively large, with the 80th quantile error is 13.0 meters and the average error is 9.1 meters, as shown in Fig. 2. Although modern smartphones have enhanced their positioning components with advanced techniques, e.g., wireless signal based mobile positioning [24], [30], we still see rather large errors as shown in Fig. 2, e.g., the average and 80th quantile errors are 7.0 meters and 11.9 meters, respectively.

Fig. 3 further demonstrates a concrete example, where we see that the trajectory observed by smartphone's GPS receiver deviates from the actual trajectory greatly, with average positioning error as large as 12.3 meters. Therefore, effective techniques are imperatively required to improve GPS accuracy in urban canyons.

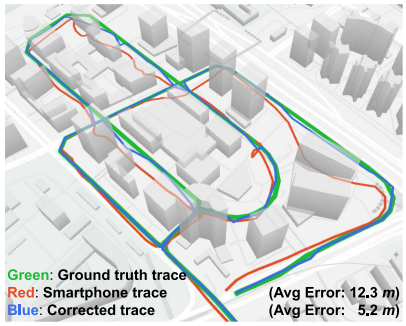

Fig. 3. Comparison on localization results of smartphones and the actual trajectory.

2.3 Motivation

The root cause of GPS errors in urban canyons is satellite signal reflection caused by complex and dynamic urban environments, in particular high-rising buildings. Thus, we imagine that there exists some relationship between GPS performance and the environment around a GPS receiver. It could be true. On one hand, for a given location p expressed by a pair of latitude and longitude, the factors of its surrounding environment that will affect satellite signals, i.e., buildings, are consistent in a long period. On the other hand, navigation satellites are constantly and regularly operating, and their positions are predicable [33]. As a result, we could enumerate all possible combinations of available satellites, including satellite identities and their positions in the sky, for location p, and for each satellite combination there will exist an observation $\hat{p}$ that is the position estimated by a GPS receiver. The relationship between actual location p and GPS estimation $\hat{p}$ can be modeled by a mapping function f, which is defined as:

$$f(p, \mathbf{c}, \mathbf{s}) \rightarrow \hat{p}, \tag{2}$$

where $\mathbf{c}$ encodes the urban environment around p, while $\mathbf{s}$ represents the satellite distribution when GPS positioning is invoked. In practice, we would like to infer ground truth position p from GPS estimation $\hat{p}$, the function is thus expressed as $f(\hat{p}, \mathbf{c}, \mathbf{s}) \rightarrow p$.

Existing works [22], [47], [58] usually assume that GPS errors, i.e., the difference between p and $\hat{p}$, follow a Gaussian distribution, while some researches [64] report that GPS errors actually follow the Raleigh distribution. In fact, both Gaussian distribution and Raleigh distribution are lack of versatility to model practical GPS estimations across a large area. For example, Wu et al. [58] have to fit one private Gaussian distribution for GPS estimations of each small road segment. These statistical models omit the important positioning contexts, e.g., buildings and satellites, and thus are inadequate to model the mapping function f.

There indeed exist a number of works that consider information of both buildings and satellites to improve GPS performance. In general, they use proprietary 3D city models to approximate function f by implicitly linking location p with visible satellites or explicitly tracing signal paths between p and observed satellites. For research works of the former category, they study and extend the idea of shadow matching [23], [49], [60], [63], [68] to improve positioning accuracy. The shadow matching algorithm determines the user's position from candidate positions with satellite visibility that compares with the GPS receiver's measurements [23]. Given the initial GPS position $\hat{p}$, a certain area around $\hat{p}$ is uniformly divided into grids, and each grid is treated as a candidate position for the actual position. For each candidate position, the algorithm predicts satellite visibility based on the 3D city model and satellites' positions. A given satellite is visible if its direct signal to the candidate position cannot be blocked by obstacles, e.g., buildings. Meanwhile, the satellite visibility can also be estimated by GPS receiver's received signals. As the reflected NLOS signal may lead to an incorrect visibility estimation, shadow matching algorithm usually assumes that the satellite with received signal strength larger than a predefined threshold is visible. Finally, the candidate position that has the best match of satellite visibility with the signal-based satellite visibility estimation is deemed as the solution of shadow matching.

Based on the precise 3D city models, research works belonging to the latter category either remedy pseudoranges by recovering "virtual direct paths" [39] (e.g., recovering actual path as the red dashed line for satellite B in Fig. 1), or compute the similarity between signal paths of candidate locations and the observed ones [29], [45]. However, both shadow matching based methods and satellite signal path based methods heavily rely on the proprietary 3D city models, which are not easily accessible. Even worse, it is difficult to properly set the threshold for determining satellite visibility in shadow matching, while recalculating pseudoranges can incur huge computation overheads that largely restrict their practicability on ordinary devices like smartphones.

Inspired by deep reinforcement learning theory that employs deep neural networks as function approximators to learn Q functions [13], we instead propose to utilize a powerful deep neural network to approximate mapping function f for better describing complex relationships between GPS estimations and positioning contexts, which include urban environment, satellite distribution, and etc. It is feasible and beneficial to build such a deep neural network model. First, thanks to the powerful representation ability of deep learning [35], we can feed coarse contextual information, which encode urban environment $\mathbf{c}$ and satellite distribution $\mathbf{s}$, rather than proprietary 3D city models, into a deep neural network model. Second, we could have abundant GPS data to well train the model. Due to the wide adoptions of GPS devices in vehicles, e.g., taxis and buses, and smart devices, e.g., smartphones, we can accumulate massive GPS measurements as the training samples [41]. Lastly, although training a deep learning model would be time-consuming, the inference is extremely quick [53]. Compared to previous works [23], [39], [45], [49] that require large storage for 3D city models and huge real-time computations, deep learning based solutions can train and deploy the bulky model at cloud, and quickly respond each request to fix the GPS estimation.

Challenges. Despite the promising advantages, it is non-trivial to realize the deep learning enhanced GPS positioning system, due to the following challenges.

First, multiple key factors, e.g., building heights and satellite statuses, could affect GPS performance in urban canyons, while they are in different modalities with varied

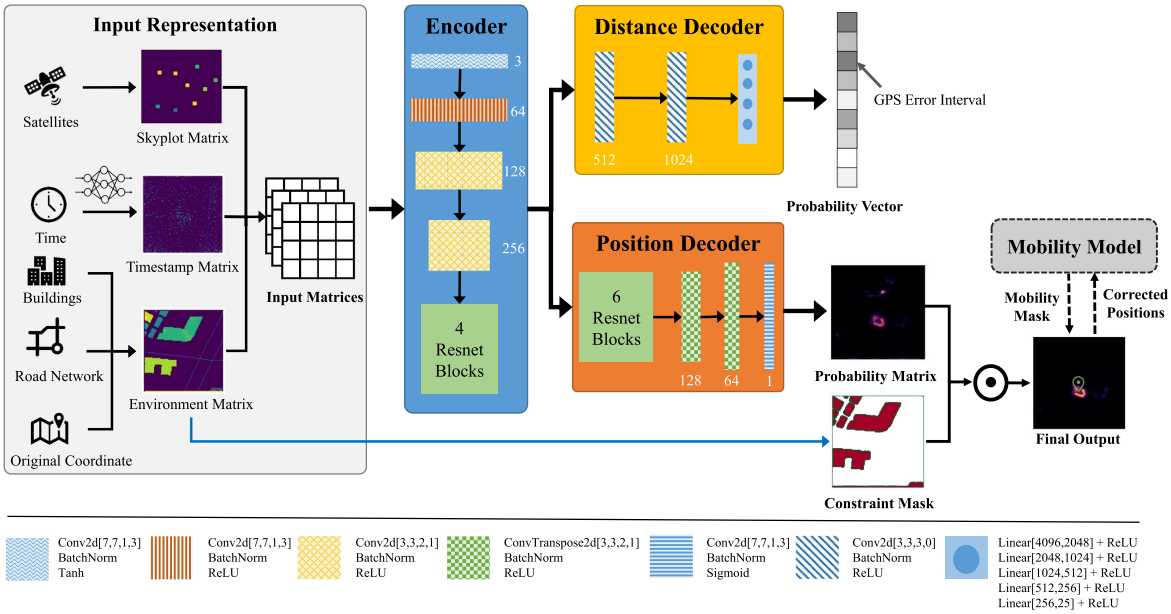

Fig. 4. The system architecture of *DeepGPS*, which takes the three representation matrices of satellite statuses, timestamp, and the surrounding environment as the input, and simultaneously predicts the positioning error interval and correct position using an encoder-decoder network model. For each Conv2d and ConvTranspose2d layer, the four key values are [*kernel-width*, *kernel-height*, *stride*, *padding*]; while for each linear layer, the two key values are [*input-dimension*, *output-dimension*]. The white number next to each block indicates the number of kernels.

dimensions. Thus, how to represent them and properly feed them into the deep neural network, without information loss, is challenging.

Second, position is normally expressed as a pair of latitude and longitude, which are hard to predict due to the huge search space. Therefore, how to devise the network architecture, including the forms of both input and target, and exploit more opportunities to refine the search space remain to be explored.

Third, location-based applications generally require to track objects of interest. Although we can invoke the model to fix each individual GPS estimation, however, such a method is inefficient and possibly with low accuracy. Therefore, how to make the deep neural network support continuous localization is a challenge.

3 Design of *DeepGPS*

In this section, we will elaborate the design details of *DeepGPS*, and discuss how to extend *DeepGPS* for continuous localization.

3.1 System Overview

Fig. 4 illustrates the system architecture of *DeepGPS*, which consists of four major modules: *Input Representation*, *Encoder*, *Distance Decoder*, and *Position Decoder*. At the high-level, *DeepGPS* fuses multiple factors that affect GPS positioning in urban canyons, and represents them in the same size of matrices. Based on the encoder-decoder network design, *DeepGPS* extracts latent features from the input matrices, and then simultaneously predicts positioning error and correct position for a given GPS estimation.

Specifically, *Input Representation* module (Section 3.2) takes satellite statuses, positioning time, and the surrounding environment (i.e., information of roads and buildings around original coordinate outputted by the GPS receiver) into account, and represents them as matrices. *DeepGPS* adopts an encoder-decoder network model to realize the deep neural network for approximating the mapping function f (Section 3.3). The *Encoder* module aims to extract high-level features from input matrices, while *Distance Decoder* and *Position Decoder* modules use derived feature maps to generate distance probability vector and position probability matrix, respectively. The distance probability vector indicates possible positioning error while position probability matrix indicates the correct position for current GPS positioning instance. Moreover, *DeepGPS* builds a constraint mask from the environment matrix to filter out impossible locations from position probability matrix, and outputs the final matrix, where the position with the largest value is the final solution. In addition, *DeepGPS* maintains a mobility model that is learned from the user's latest movements, and exploits this model to refine the output for efficient continuous localization (Section 3.4).

3.2 Multi-Source Data Fused Input Representation

Besides the inconsistencies of atmospheric conditions and Doppler effect, multiple factors may affect GPS performance in the urban canyons, which should be considered as the input for *DeepGPS*. As aforementioned in Section 2.3, GPS errors in urban canyons are primarily caused by the satellite signal reflections. In principle, whether a single is reflected or not is determined by both satellite position and heights of buildings[2] around the GPS receiver. In practice, people usually walk or drive on roads in urban areas, thus road information are important to restrict users' possible locations. Besides, the specific time when GPS positioning instance is taken should also be considered for the following reasons. First, human mobility is regular and consistent [66], and thus time can be an implicit indicator of urban dynamics that may influence

2. We omit tall trees and other high-rising things, since they are less common in the GPS positioning scenarios.

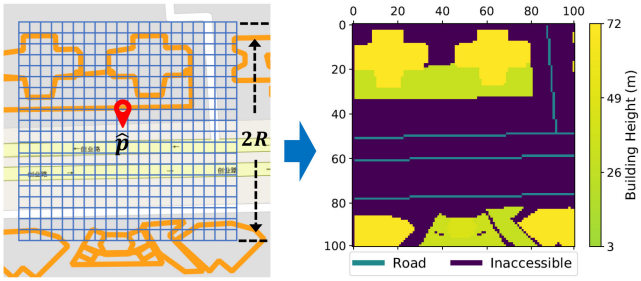

Fig. 5. (a) The square area of interest given the estimated position $\hat{p}$ and maximum error distance R; (b) The environment matrix $\mathbf{M}_e$ represents environmental factors within the square area in (a).

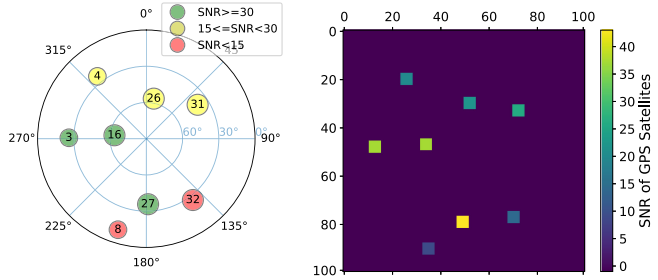

Fig. 6. (a) An example skyplot of satellites seen by the receiver; (b) The skyplot matrix $\mathbf{M}_s$ transformed from the skyplot of (a).

GPS positioning. Second, navigation satellites are regularly operating and time could be used to indirectly encode the information of satellite distribution as well. In summary, we take satellite statuses, time, and surrounding environment as the input of *DeepGPS*. In particular, we describe the environment using information of roads and buildings near the original coordinate provided by the receiver.

Instead of separately building a deep neural network model for each input data source and strategically aggregating outputs of all models to derive the final result, we prefer to fuse multi-source data as the input. Such an operation can avoid information loss of raw data and the troublesome training of multiple models. To this end, we represent multi-source data in the same size of matrices, which can well capture spatial relations among objects of interest. We detail the representation of each data source as follows.

(1) *Representation of surrounding environment* When positioning is required, the GPS receiver will provide an estimated position $\hat{p}$ in the form of *[latitude, longitude]* and an error that specifies position uncertainty. In principle, the actual position p is within a circle centered at estimated position $\hat{p}$ with its radius as the GPS error. Thus, we consider the environment, especially the height and layout of buildings and the geographic distribution of roads, around $\hat{p}$ to search for the actual position p.

We construct an *environment matrix* $\mathbf{M}_e$ to represent these environmental factors that may affect this GPS positioning instance. We first select a square area, which centers at $\hat{p}$ with side length as $2R$, and then divide this area into cells with size of $c \times c$. These cells are treated as candidate locations where p may locate. Each cell in the square area corresponds to an element of matrix $\mathbf{M}_e$. For each element in the matrix, its value is set according to the following rules: *i)* If the cell is part of a building, its value is set as the building height; *ii)* If the cell is part of a road, the value is zero; *iii)* Otherwise, it is -1, which implies the corresponding cell is inaccessible. According to our analysis on massive GPS data, we conservatively set $R = 50$ meters, which is larger than 99% GPS errors as shown in Fig. 2. In addition, we set the cell size $c = 1$ meter to achieve fine-grained position fixes. Therefore, matrix $\mathbf{M}_e$ is of size 100×100. Fig. 5a demonstrates how we divide the area of interest into candidate cells, and construct the corresponding environment matrix $\mathbf{M}_e$ as shown in 5(b).

(2) *Representation of satellite statuses* In addition to the estimated position $\hat{p}$, we can obtain satellite metadata from the receiver as well, which include each satellite's azimuth, elevation, and signal-to-noise ratio (i.e., SNR). Given the received satellite metadata, a *skyplot* can be draw to illustrate the satellite geometry over a given ground site [43]. Fig. 6a shows an example skyplot that is plotted with satellite metadata collected during our real-field experiments. Each circle represents a satellite detected by the GPS receiver, with the number as satellite identity. The color of each circle indicates the satellite's signal strength: green is very good, yellow is fair, while red is bad.

Skyplot is an effective representation of satellite statuses, and previous works usually exploit skyplot to filter out NLOS satellites [50], [60], or perform shadow matching based position fixing [23], [49], [68], with assistance of 3D city models. Thus, we attempt to transform the skyplot into a matrix as partial input of *DeepGPS*. We call this matrix as the *skyplot matrix* $\mathbf{M}_s$ because it can encode relative positions and SNR values of all detected satellites. We set matrix $\mathbf{M}_s$ as the same size as $\mathbf{M}_e$, and embed information of each satellite on the skyplot into matrix $\mathbf{M}_s$. First, we align the centers of skyplot and $\mathbf{M}_s$. Then, for each detected satellite, its position is mapped from the skyplot to $\mathbf{M}_s$ leveraging satellite's elevation and azimuth. Specifically, the satellite's elevation determines the distance between its position on matrix $\mathbf{M}_s$ to $\mathbf{M}_s$'s center, while satellite's azimuth determines the angle clockwise-positive from the "up" direction. If the position on matrix $\mathbf{M}_s$ "has" a satellite, its value is set as the corresponding satellite's SNR value; otherwise, its value is zero. Fig. 6b shows the skyplot matrix for the example skyplot in Fig. 6a.

(3) *Representation of timestamp* Ingeneral, timestamp of a GPS positioning instance is represented as a number. According to the operation rules of GPS satellites, each satellite orbits the Earth every 12 hours, which means that it will return back to the same place periodically in theory. However, since the Earth rotates at the same time, the relative position from satellites to the receiver at the same location may be different after one orbital period. Recent studies [11], [61] report that the revisit period is variable across the GPS satellites. Specifically, the revisit period of GPS satellites is slightly different and a bit earlier than a day for each satellite in the range of $240\,s$ and $250\,s$. In addition, the average satellite revisit period is $246\,s$ less than a day [11]. For simplicity, we set the revisit period for all satellites as $86154\,s$ (i.e., $24 \times 3600 - 246$)[3], which means the receiver at a

3. Even the revisit period for the same GPS satellite is changing across time [61], and the accurate calculation requires extra information. We thus leave the dynamical revisit period calculation for each GPS satellite as a future work.

TABLE 1
The Operations Applied to Calculate Each Element in the Vector $\mathbf{V}_t$ from Timestamp T

Element	Description
$\mathbf{V}_t[0]$	$(t\%86400)/86400$
$\mathbf{V}_t[1]$	$(t\%86154)/86154$
$\mathbf{V}_t[2]$	$\cos(2 * \pi * \mathbf{V}_t[0])$
$\mathbf{V}_t[3]$	$\sin(2 * \pi * \mathbf{V}_t[0])$
$\mathbf{V}_t[4]$	$\cos(2 * \pi * \mathbf{V}_t[1])$
$\mathbf{V}_t[5]$	$\sin(2 * \pi * \mathbf{V}_t[1])$
$\mathbf{V}_t[6]$	Before noon $(= 1)$ or after noon$(= 0)$

fixed location will "see" the same GPS satellites again after such a revisit period. Based on the revisit periodicity of GPS satellites, we generate a timestamp matrix $\mathbf{M}_t$ from timestamp t.

First, we convert the given timestamp t into a 7-dimension vector $\mathbf{V}_t$ by considering the prior knowledge about revisit period and rotation period. Each element of vector $\mathbf{V}_t$ is derived from the specific operation as shown in Table 1. Specifically, element $\mathbf{V}_t[0]$ and $\mathbf{V}_t[1]$ represent the progress in a rotation period of the Earth and in a revisit period of satellites, respectively. Noting that the total seconds of a revisit period is set as 86154 seconds, and the total seconds of a nature day (i.e., 24 hours) is 86400 seconds. The 3rd to 6th elements of vector $\mathbf{V}_t$ are the results of applying *Sine* and *Cosine* to $\mathbf{V}_t[0]$ and $\mathbf{V}_t[1]$, which can help to quickly find the periodicity [10]. In addition, element $\mathbf{V}[6]$ is set as one if t is before noon of a day; otherwise zero.

Then, we apply a simple multilayer perception (MLP) model to transform vector $\mathbf{V}_t$ to an embedding. The MLP model consists of five layers, including one input layer, three hidden layers, and one output layer. Fig. 7 illustrates the MLP model structure. Specifically, the input layer takes the 7-dimension vector $\mathbf{V}_t$ as the input; the three hidden layers have 100, 1000, and 2000 nodes, respectively; while the output layer will generate an embedding of size 10000. In addition, we adopt the rectified linear unit (ReLU) as the activation function. The derived embedding is reshaped into a matrix having the same size 100×100 as $\mathbf{M}_e$ and $\mathbf{M}_s$. We call this matrix as *timestamp matrix* $\mathbf{M}_t$ for the given timestamp t, which implies the periodical features of the GPS positioning time.

3.3 Deep Neural Network Design

Given these input matrices, one may build a deep neural network to directly predict the correct position. However, the searching space is infinite as both latitude and longitude are float numbers, which make the derived model hard to train and use in practice. Instead, we set the target output as a matrix with the same size as input matrices, and configure the desired model to predict which cell correct position locates. The center of predicted cell is treated as the fixed position. In the meanwhile, we actually do not need to correct all positioning results, since many GPS estimations may be sufficiently accurate for upper applications. We thus expect that our model can predict the positioning error as well, then we could determine whether to fix current GPS

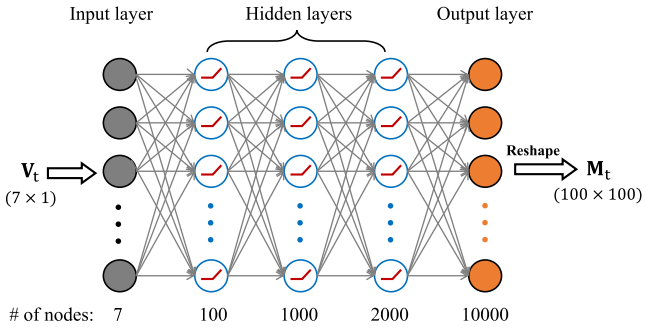

Fig. 7. The MLP model structure.

estimation according to the requirements of location-based applications.

To that end, we model the desired deep neural network as an encoder-decoder architecture, where two decoders feed off the encoder, as shown in Fig. 4. Specifically, encoder $\mathtt{E}$ learns a feature map $\mathcal{M}$ from input matrices (i.e., $\mathbf{M}_e$, $\mathbf{M}_s$, and $\mathbf{M}_t$), and feeds $\mathcal{M}$ to distance decoder $\mathtt{D}_{dist}$ and position decoder $\mathtt{D}_{posi}$ to predict positioning error and correct position, respectively. An extra advantage of the dual-decoder design is that given the same feature map $\mathcal{M}$, position decoder and distance decoder can mutually bound each other and make their predictions best match with the receiver's observations and positioning context.

Model structure. Since the input of our model are matrices, which are similar as images, we thus construct encoder $\mathtt{E}$ with a series of 2D convolution layers to extract spatial features from the positioning contexts that are represented by environment matrix, skyplot matrix, and timestamp matrix. In addition, we adopt 4 and 6 Resnet blocks in encoder $\mathtt{E}$ and position decoder $\mathtt{D}_{posi}$, respectively, to form a deep neural network which can explore enough feature space and avoid vanishing gradient, simultaneously. The implementation of Resnet blocks is inspired from [32].

Distance decoder: We put forward distance decoder, $\mathtt{D}_{dist} : \mathcal{M} \rightarrow \mathbf{V}_{dist}$, to predict positioning error given feature map $\mathcal{M}$. Decoder $\mathtt{D}_{dist}$ applies two 2D convolution layers to process $\mathcal{M}$, then the intermediate result is flattened and fed into four linear layers to derive final output $\mathbf{V}_{dist}$. Rather than predicting an exact error, which can be modeled as a regression problem, we prefer to predict an error interval, where the exact error falls into. Thus, we define output $\mathbf{V}_{dist}$ as a probability vector, where each element of $\mathbf{V}_{dist}$ corresponds to an error interval and its value indicates a probability. In our implementation, we set the interval as 2 meters and $\mathbf{V}_{dist}$ with size of 25-dimension, as GPS errors over 99% cases are smaller than 50 meters. Thus, the i-th element of $\mathbf{V}_{dist}$ means that the error is in the range of $(2 * i, 2 * (i + 1)]$ meters.

Position decoder: Similarly, position decoder, $\mathtt{D}_{posi} : \mathcal{M} \rightarrow \mathbf{R}_{posi}$, is devised to predict correct position p' from the feature map $\mathcal{M}$. As shown in Fig. 4, $\mathtt{D}_{posi}$ shares a similar but inverse structure as encoder $\mathtt{E}$, which firstly applies 6 Resnet blocks to feature map $\mathcal{M}$, then two 2D transposed convolution layers are used to re-scale the intermediate result, ended with one 2D convolution layer. The output $\mathbf{R}_{posi}$ is a matrix of the same size as input matrices, and thus each element in $\mathbf{R}_{posi}$ corresponds to a candidate cell defined in environment matrix $\mathbf{M}_e$. Instead of marking the cell containing the ground truth position as one and the rest zeros, we set the

target position as a Gaussian peak. Therefore, $\mathbf{R}_{posi}$ is a probability matrix, and the element with the largest probability implies where the ground truth position locates. Compared to one-hot encoding, Gaussian peak representation can help to avoid gradient vanishing during the model training [14].

Constraint mask: The position decoder $\mathtt{D}_{posi}$ may predict any cell as the target, however, some cells that are occupied by buildings or obstacles obviously cannot contain correct position. Therefore, we propose the constraint mask $\mathcal{C}_{env}$, which embeds prior knowledge of surrounding environment, to constrain the output of $\mathtt{D}_{posi}$. Specifically, the element of $\mathcal{C}_{env}$ is set as zero if the corresponding cell is inaccessible; otherwise the element is one and the corresponding cell is a valid candidate.

The blue flow in Fig. 4 demonstrates how we make use of constraint mask $\mathcal{C}_{env}$ to enhance position fixing. The final output $\mathbf{R}_{env}$ is the element-wise product of original output $\mathbf{R}_{posi}$ of $\mathtt{D}_{posi}$ and constraint mask $\mathcal{C}_{env}$, i.e., $\mathbf{R}_{env} = \mathbf{R}_{posi} \odot \mathcal{C}_{env}$. With these operations, the inaccessible cells can be filtered out, while the cell with the largest value in $\mathbf{R}_{env}$ is considered as the final cell containing the correct position.

Loss Function. We adopt one-hot encoding to prepare target vector $\mathbf{V}_{true}$ for distance decoder $\mathtt{D}_{dist}$. For each GPS sample, we calculate its true positioning error e using the GPS estimation and actual position, and then compute the error interval $j = \lfloor \frac{e}{2} \rfloor$ where e falls into. Then, we prepare target vector $\mathbf{V}_{true}$ for this GPS sample by setting the value of $\mathbf{V}_{true}$'s j-th element as *one* and the values of the rest elements as *zero*. Because we model positioning error prediction as a classification problem, we thus adopt cross-entropy loss to measure the distance between $\mathtt{D}_{dist}$'s output $\mathbf{V}_{dist}$ and the target $\mathbf{V}_{true}$, which is defined as:

$$\mathcal{L}_{dist} = -\sum_{i=1}^{n} b_i \log(p_i), \tag{3}$$

where b_i is a binary indicator ($b_i = 1$ if the i-th interval is the truth; otherwise $b_i = 0$), p_i is a Softmax probability for the i-th interval, and n is the size of $\mathbf{V}_{dist}$. By default, we set $n = 25$.

For position decoder, we instead utilize the mean-square error loss (i.e., L2-norm loss) to measure distance between the enhanced output $\mathbf{R}_{env}$ and the target $\mathbf{R}_{true}$, which is the Gaussian representation of ground truth position. Therefore, the loss function for position decoder $\mathtt{D}_{posi}$ is defined as:

$$\mathcal{L}_{posi} = ||\mathbf{R}_{true} - \mathbf{R}_{env}||^2, \tag{4}$$

where $||\cdot||^2$ is the L2-norm loss.

Finally, the overall loss function for our model is a weighted sum of both $\mathcal{L}_{dist}$ and $\mathcal{L}_{posi}$, i.e.,

$$\mathcal{L}_{overall} = \lambda \times \mathcal{L}_{dist} + \mathcal{L}_{posi}, \tag{5}$$

where λ is a regularization parameter to balance the influences of distance decoder and position decoder on the encoder. In order to determine the proper setting of λ, we separately train $\mathtt{D}_{dist}$ and $\mathtt{D}_{posi}$ to observe their losses, and then set λ to make the losses of the two decoders well-balanced. According to our experiments, we finally set $\lambda = 0.001$, which can achieve the best prediction performance for both decoders.

Model Training. We utilize massive labeled GPS samples, along with road network, building survey data, and satellite data, to train *DeepGPS*. For each GPS positioning instance and its ground truth position, we build an environment matrix $\mathbf{M}_e$ by exploiting the information of road network and building heights, a timestamp matrix $\mathbf{M}_t$ from the positioning time, and a satellite matrix $\mathbf{M}_s$ from the corresponding skyplot. In addition, we construct a vector $\mathbf{V}_{true}$ of 25-dimension via one-hot encoding and a matrix $\mathbf{R}_{true}$ that marks the cell containing actual position as the Gaussian peak. The vector $\mathbf{V}_{true}$ and matrix $\mathbf{R}_{true}$ are target outputs of distance decoder $\mathtt{D}_{dist}$ and position decoder $\mathtt{D}_{posi}$, respectively.

With the proper setting of λ and the loss function $\mathcal{L}_{overall}$, we train the network holistically using the labeled GPS samples. Since the distance loss and position loss add up to update the common encoder, the two decoders can mutually access information from each other and, as a result, achieve more accurate predictions. It is worthy to note that we treat the MLP model, which transforms time t to a timestamp matrix $\mathbf{M}_t$ as part of encoder $\mathtt{E}$, and thus we train MLP model along with the encoder.

3.4 Extend to Continuous Localization

With sufficient positioning instances, user's mobility information can be inferred, which can be utilized to further improve position fixing by reducing location uncertainty. Specifically, *DeepGPS*maintains a mobility model, as illustrated in Fig. 4, which aims to roughly estimate user's moving speed and utilizes the speed to calculate a reachable area to constrain future localization.

*Mobility Model. DeepGPS*makes use of the latest k corrected positioning instances, in particular the positions $(p'_{i-1}, \cdots, p'_{i-k})$ and the corresponding positioning time $(t_{i-1}, \cdots, t_{i-k})$, to infer user's movement. For simplicity and generality, *DeepGPS*only estimates the user's average moving speed v, because other mobility information, e.g., moving direction, require extra sensor data. For a given user, *DeepGPS*continuously updates the average speed v with the latest k positioning instances through the follow equation:

$$v = \frac{\sum_{j=1}^{k-1} \frac{L(p'_{i-j}, p'_{i-j-1})}{t_{i-j} - t_{i-j-1}}}{k-1}, \tag{6}$$

where $L(p'_{i-j}, p'_{i-j-1})$ calculates the distance between position p'_{i-j} and p'_{i-j-1}. When there are insufficient positions for speed calculation, we set k as the available number for the initial use.

Mobility Improved Position Fixing. Once the average moving speed v is ready, we can derive an extra mobility constraint for position correction. For GPS estimated position $\hat{p}_i$, on one hand, we construct a square area centered at $\hat{p}_i$ with side length of $2R$, and divide the square area into cells with size of $c \times c$. On the other hand, we construct a circle, which is centered at position p'_{i-1}, i.e., the corrected position of last positioning instance $\hat{p}_{i-1}$, with radius of $S = v \times \Delta t$, where Δt is the time difference between positioning time t_i of $\hat{p}_i$ and time t_{t-1} of $\hat{p}_{i-1}$. The circle area indicates the reachable area of the user according to her recent moving speed. Combining GPS position and reachable area, cells in the intersection of square area and circle area are the most likely

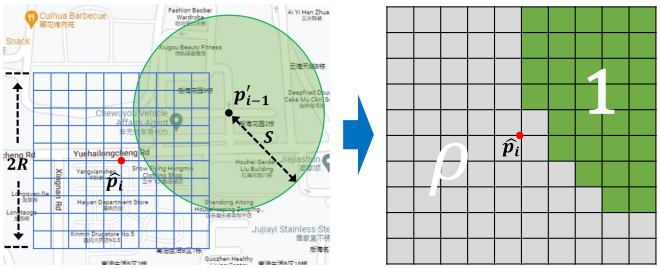

Fig. 8. (a) The candidate cells of GPS estimation $\hat{p}_i$ intersect with the reachable area from position p'_{i-1}; (b) The mobility mask for $\hat{p}_i$, where green cells are reachable with confidence 1 while gray cells may be reachable with a lower confidence ρ.

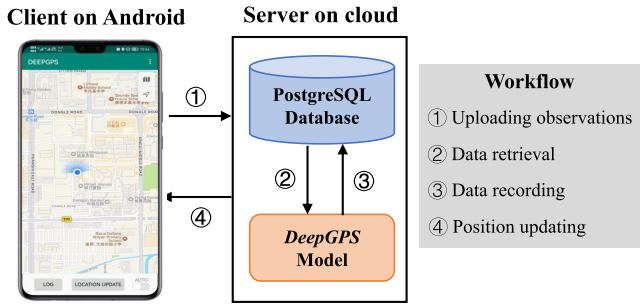

Fig. 9. The implementation and workflow of *DeepGPS*.

candidates to contain correct position of $\hat{p}_i$, while other cells of the square area are less possible. Based on this intuition, we construct a matrix named as *mobility mask* $\mathcal{C}_{mob}$, each element of which corresponds to a cell in the square area. Specifically, if the cell is (or partially) covered by the circle, its value is set as one; otherwise we set its value as ρ $(0 \leq \rho < 1)$, which means this cell may contain correct position with confidence ρ. It is worthy to note that if we set $\rho = 1$, we actually disable the function of mobility improved position fixing. Fig. 8 illustrates an example that makes use of both mobility information and GPS estimation to determine the mobility mask.

Instead of applying constraint mask and mobility mask together to the original output of position decoder, mobility mask will take effect after constraint mask. The reason behind is that we can make the mobility model be independent of deep neural network, and thus alleviate the model training overheads. Therefore, we dot-multiply mobility mask $\mathcal{C}_{mob}$ with $\mathbf{R}_{env}$, which is the processed result of applying constraint mask to original output of position decoder, and then derive the final probability matrix $\mathbf{R}_{mob} = \mathbf{R}_{env} \odot \mathcal{C}_{mob}$, which is rectified according to mobility information. Finally, the center of the cell owing the largest value in $\mathbf{R}_{mob}$ is outputted as the corrected position of *DeepGPS*.

In fact, our mobility model can be further enhanced with richer mobility data and advanced tracking techniques. For example, if inertial measurement unit (IMU) sensor data of the user's smartphone are available, we could derive more mobility information about the user [62], e.g., moving speed and direction, and exploit more advanced techniques, e.g., particle filter [59], to track user's movements, and thus refine the reachable area to greatly reduce location uncertainty. We leave this study as our future work.

4 Implementation

We design and implement a prototype on the cloud (as the server) and several Android smartphones (as the clients), as illustrated in Fig. 9. *Android clients* collects and uploads raw GPS measurements to the *cloud server* that is responsible for predicting error distance and correct position. We detail the implementation of each component and the whole workflow as follows.

Client on the Android. We implement the client component on smartphones with Android OS, which allows us to easily access the raw GPS measurements (including GPS estimated position and error, satellite metadata, etc.) through Android APIs [2]. The client can log raw GPS measurements for data analysis. If the user needs more accurate positioning in urban canyons, she can trigger the "*location update*" button to fix GPS estimation, which is supported by *DeepGPS*model on the cloud. If necessary, the user can also enable automatic location updating function by switching on the "*auto*" option, which will allow *DeepGPS*to analyze user's coarse mobility information for continuous localization.

Server on the Cloud. We implement *DeepGPS*model in PyTorch 1.8.1 (CUDA 11.1) [7] on a server with CPU of Intel (R) Core(TM) i7-10700K 3.80GHz, GPU of RTX3090, and memory of 48GB. To train our model, we use Adam as the optimizer, and set learning rate $\alpha = 1e-5$ and batch size as 128. Furthermore, we deploy *DeepGPS*model on the cloud, so that users can access to the position fixing service anytime and anywhere. The cloud deployment will also alleviate the computation and storage overheads for the smartphone clients [37].

At the server side, we also maintain a spatial database, which is implemented in PostgreSQL [6] with a spatial database extender – PostGIS[5] for efficient data querying. The spatial database is used to store road network and building survey data (e.g., height and boundary of buildings) of the testing city.

The spatial database interacts with *DeepGPS*model for data retrieval and recording. On one hand, given the estimated position $\hat{p}$ from a client, *DeepGPS*can retrieve road network and building information around $\hat{p}$ to construct the environment matrix. On the other hand, when a user enables continuous localization, *DeepGPS*will record the user's historical positioning data into the database for dynamically updating this user's mobility model.

Workflow. As illustrated in Fig. 9, *DeepGPS*performs the position fixing upon each request as follows.

① When accurate positioning is required, the client firstly acquires GPS estimated position $\hat{p}$ and other satellite metadata via Android APIs, and then communicates these data (including $\hat{p}$, timestamp t, and satellite metadata) and an error threshold δ to the server. Noting that threshold δ is optional, and the users can let *DeepGPS*always correct GPS estimations. If necessary, threshold δ could be specified by some location-based application according to its requirements on positioning accuracy.

② After receiving the request, *DeepGPS*model retrieves data of road network and buildings around position $\hat{p}$ from the spatial database, and then constructs environment matrix $\mathbf{M}_e$, timestamp matrix $\mathbf{M}_t$, and satellite matrix $\mathbf{M}_s$,

respectively, by exploiting the multi-source data. Once these input matrices are ready and if δ is provided, the model will firstly exploit distance decoder $\mathtt{D}_{dist}$ to predict possible error e. If e is smaller than threshold δ, *DeepGPS*treats $\hat{p}$ as the correct position p', since $\hat{p}$ is sufficiently accurate for the user's applications. Otherwise, *DeepGPS*invokes position decoder $\mathtt{D}_{posi}$ to predict correct position p'. Noting that *DeepGPS*offers the best-effort service only. As we have no way to know the ground truth position, *DeepGPS*has no further operation after predicting the correct position. If δ is not provided, $\mathtt{D}_{posi}$ is directly executed to infer p'.

③ Once the positioning request has been processed, *DeepGPS*will log a record $< t, \hat{p}, p' >$ to the database. This record indicates that the requester visited position p' at time t, and will be appended to a specific file of the requester. These records help *DeepGPS*to dynamically update the given user's mobility model.

④ Finally, the server sends back position p' to the client.

5 Performance Evaluation

In this section, we evaluate the performance of *DeepGPS*with a large bus GPS trajectory dataset and real-field GPS measurements that are collected using our Android clients.

5.1 Experiment Setup

We conduct all experiments in Shenzhen city, China, which has the second most skyscrapers in the world [9].

Dataset. We collect five different types of data for the performance evaluation. Specifically, road network and building survey data are used to represent the positioning environment, and satellite data describe the satellite distribution and statuses. In addition, we treat both bus trajectory data and real-field GPS measurements as positioning instances for model training and testing.

Road Network. We download the road network of Shenzhen city from OpenStreetMap (OSM)[4]. The OSM file contains all roads and points-of-interest (POIs), e.g., lake and grassland, of our testing city in the form of *nodes* (i.e., points), *ways* (i.e., roads), and *relations* (i.e., properties of POIs). In particular, we can exploit relations to distinguish whether a given cell is accessible or not.

Building Survey Data. We obtain the building survey data from our collaborators. This survey file shares the similar format as OSM file, and contains information about the layouts, heights, and properties of all buildings in Shenzhen city. Specifically, a list of *nodes* are connected end to end to form the outline of each building, while the height and building properties (e.g., name) are tagged in *relations*. These data help us to identify the building heights and inaccessible areas in a city.

Bus Trajectory Data. Equipped with GPS devices, public buses can periodically report their statuses back to the operation center. In general, the buses in Shenzhen city operate regularly following fixed routes and schedules, and send back a report every 5 seconds [56]. Each report includes a timestamp, GPS location, travel speed, direction, status, and etc. We prepare a bus GPS trajectory dataset that was collected by 16690 buses covering 1845 routes on June 12th, 2020. These bus routes cover most urban areas of Shenzhen city. In total, we have 41540968 bus reports.

Real-Field GPS Measurements. By installing our Android client (see Section 4) on their smartphones, five volunteers drive private vehicles to collect GPS measurements every 5 seconds, following some planned routes in the downtown area of Nanshan District, Shenzhen city. Their smartphones include HUAWEI Mate 10 pro, Mate 30 pro, Mate 40 pro, and Samsung Galaxy Note 5. Finally, we have collected 16814 valid positioning samples.

Satellite Data. Our Android client can collect satellite metadata, and thus the real-field GPS dataset contains satellite information when taking a positioning sample. However, those bus reports do not include satellite metadata. To make up, we visit CelesTrak [1] to retrieve historical satellite metadata given the timestamp and actual location of each bus report. As a result, we can supplement all bus reports with the corresponding satellite data.

Ground Truth Collection. Similar as the previous works [22], [53], [58], we use an advanced hidden Markov based map matching algorithm [47] to map GPS sequences of both bus trajectories and real-field GPS measurements to road segments. Because both public buses and testing vehicles drive along the planned routes, we can exploit such prior knowledge to verify the map matching results and manually correct these erroneous matching results. Finally, for each GPS estimated position $\hat{p}$, we treat its projection on the matched road segment as the ground truth p.

Testing Regions and Model Training. Since we target on improving GPS performance in urban canyons, we thus select three downtown areas of Nanshan District, Futian District, and Baoan District, respectively, in Shenzhen city as our testing regions. The three regions own the densest high buildings of Shenzhen city, and we denote them as *region N*, *region F*, and *region B*, respectively. We collect real-field GPS measurements in *region N* as well.

We keep bus reports that fall within the three regions for experiments. Considering that different regions have distinct environments, we thus train a specialized *DeepGPS*model for each region using its own bus reports. For each region, we utilize 70% of bus reports to train its model, while keeping the rest 30% for the testing. All real-field GPS data are used for testing only. Besides, we will compare the performance of a unified model that is trained using data of the three regions with these customized models later.

Performance Metrics. We define the following three metrics to evaluate *DeepGPS*'s performance.

- *Accuracy*. Prediction accuracy is measured as the average distance between correct position outputted by the model and ground truth, i.e.,

$$accuracy = \frac{1}{N}\sum_{i=1}^{N} L(p_i, p_i'), \tag{7}$$

  where N is the number of GPS samples, p_i and p_i' are the i-th ground truth and correct position, respectively, while function $L(p_i, p_i')$ returns a distance between them.
- *Effective ratio*. In addition to *accuracy*, we define effective ratio to evaluate the positioning improvement of *DeepGPS*over the original GPS estimations. Specifically, we define effective ratio as the proportion of

TABLE 2
Average Accuracy (In *Meters*) of Models Applied Across Regions

Testing Data	Model				
	Unified	*Region N*	*Region F*	*Region B*	*None*
Region N	4.2	**3.6**	5.3	6.8	8.5
Region F	4.1	4.8	**2.3**	6.5	7.2
Region B	3.2	5.9	5.2	**2.3**	7.2

The last column reports average GPS error of each region.

samples whose correct positions predicted by *DeepGPS*are closer to the ground truths than GPS's estimated positions, i.e.,

$$ratio = \frac{\sum_{i=1}^{N} \mathbb{I}(L(p_i, p_i') < L(p_i, \hat{p}_i))}{N} \times 100\%, \quad (8)$$

where indicator function $\mathbb{I}(a)$ will return one if the condition a is true; Otherwise zero.

- *Prediction error*. Since our model can predict positioning error distance, we thus define the average prediction error to evaluate *DeepGPS*as follows:

$$error = \frac{\sum_{i=1}^{N} |\arg\max(\mathbf{V}_{dist}^{i}) - \arg\max(\mathbf{V}_{true}^{i})|}{N} \times 2, \quad (9)$$

where $\arg\max(\mathbf{V})$ returns the index of the largest element in vector $\mathbf{V}$. Recall that each interval in vector $\mathbf{V}$ corresponds to 2 meters, thus the average index offset is multiplied by 2 to derive the prediction error.

We use accuracy and effective ratio to evaluate the effectiveness of position decoder $\mathtt{D}_{posi}$, while exploiting prediction error to evaluate distance decoder $\mathtt{D}_{dist}$. In the following experiments, we set $\lambda = 0.001$ and $\delta = 0$ that enforces position decoder to correct each GPS estimation. By default, we set $k = 5$ and $\rho = 0.4$ for the mobility model, and set $c = 1$ meter for the cell size.

5.2 Evaluation on Bus Data

Overall Performance. For each testing region, we train a customized model using GPS samples that are collected in this region. In addition, we train a unified model based on the training data of the three regions. To investigate the generalization ability of *DeepGPS*, we also apply one region's model to other two regions for cross-validations. We have disabled the mobility model for experiments in this subsection.

As shown in Table 2, we find the unified model achieves high positioning accuracy for the three regions, with accuracy as 4.2 meters, 4.1 meters, and 3.2 meters, respectively. Compared to the unified model, the customized models derive much better accuracy. Specifically, each customized model achieves the best accuracy on its own region, e.g., the best accuracy results of models trained on datasets of *region N*, *region F*, and *region B*, are 3.6 meters, 2.3 meters, and 2.3 meters, respectively. While we still observe relatively high accuracy for cross-region applications. For example, the model trained for *region N* achieves average accuracy as 4.8 meters on testing data of *region F*, which is much better than average GPS error of *region F*, i.e., 7.2 meters. These results imply that the *DeepGPS*model has good generality and can achieve the best performance if it is specially trained with samples of the target region. Compared to original GPS errors, *DeepGPS*improves GPS positioning by 57.6%, 68.1%, and 68.1% for *region N*, *region F*, and *region B*, respectively, with an average as 64.6%.

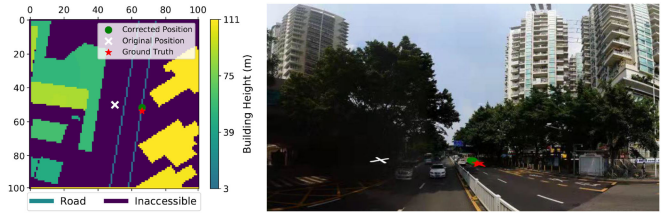

Fig. 10. An example of *DeepGPS*'s position fixing.

Fig. 10 demonstrates a concrete example, where *DeepGPS*correctly fixes the GPS estimation. Due to the impact of surrounding buildings, GPS mistakenly positions the user on a neighboring road segment, as shown in Fig. 10a. We further manually investigate the positioning environment, as shown in Fig. 10b, which turns to be a typical street canyon in our testing city.

We observe similar results for the metrics of effective ratio and prediction error, as shown in Tables 3 and 4, respectively. From the experiment results in Table 3, we see that although the unified model can achieve high effective rations on the three regions, i.e., with average effective ratio as 85.3%, each customized model derives the highest effective ratio on the target region, with an average effective ratio as 90.1%. Even if the customized model is trained on one region's samples and is leveraged to correct GPS estimations of other regions, *DeepGPS*still performs well with effective ratio larger than 72.0%. The results in Table 3 indicates that *DeepGPS*can derive a better corrected position than GPS's original output in the majority of cases.

Table 4 presents the evaluation results of *DeepGPS*'s distance decoder $\mathtt{D}_{dist}$. We find that the prediction errors are quite small, i.e., $1.4 \sim 4.1$ meters. In particular, if we train and test *DeepGPS*model on the same region, the average prediction error is only 1.7 meters, which corresponds to one interval offset only in the 25-dimension vector $\mathbf{V}_{dist}$. Thus, $\mathtt{D}_{dist}$ performs quite good.

According to above experiment results, we find that *DeepGPS*has excellent generality and positioning performance. On average, *DeepGPS*improves GPS positioning accuracy by 64.6%, and can effectively correct GPS estimations in 90.1% cases.

Effect of Building Heights. We explore the effect of building heights on positioning performance in urban canyons by exploiting environment matrix $\mathbf{M}_e$ that contains building height information. For each GPS sample, we calculate building height around the GPS receiver as the average of elements in $\mathbf{M}_e$, whose values are greater than zero. Fig. 11 compares the positioning accuracy of both GPS and *DeepGPS*under various building heights. Noting that each value x of X-axis indicates a range of building heights, i.e., $[x, x + 20)$ meters. Typically, higher buildings are more likely to reflect or even block the satellite signals, and thus significantly affect positioning. The accuracy results of both GPS and *DeepGPS*become worse when there are higher buildings around the receiver. However, *DeepGPS*still performs better than GPS, with much smaller positioning errors.

TABLE 3
Effective Ratio of Models Applied Across Regions

Testing Data	Model			
	Unified	*Region N*	*Region F*	*Region B*
Region N	83.7%	**85.6%**	78.1%	75.0%
Region F	84.4%	78.0%	**91.5%**	72.1%
Region B	87.9%	74.1%	78.9%	**93.3%**

TABLE 4
Prediction Errors (In *Meters*) Across Regions

Testing Data	Model			
	Unified	*Region N*	*Region F*	*Region B*
Region N	3.1	**1.4**	3.3	4.0
Region F	2.8	3.1	**1.9**	3.4
Region B	2.0	4.1	2.5	**1.8**

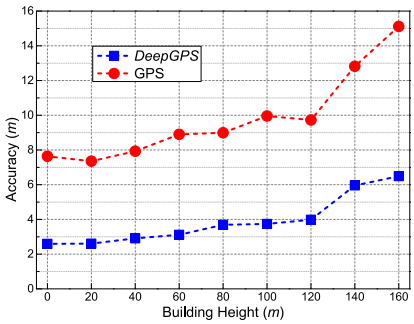

Fig. 11. Effect of building heights on the accuracy of GPS and *DeepGPS*.

As shown in Fig. 12, we observe that building height also affects *DeepGPS*'s performance on the metrics of effective ratio and prediction error. In general, higher buildings cause effective ratio of *DeepGPS*decrease, while increasing the prediction error.

Effect of Positioning Time. We examine the effect of time on the accuracy of GPS and *DeepGPS*as well, and plot the results in Fig. 13. We see that the accuracy of both systems slightly varies across time of day. *DeepGPS*still works much better than GPS over all the time. In addition, we observe similar varying trends for the two metrics of effective ratio and prediction error in Fig. 15. From these two figures, we observe slight performance degradation of *DeepGPS*in some hours, e.g., 11AM and 11PM. However, the reason needs to be further explored.

5.3 Evaluation on Real-Field Data

In this subsection, we make use of real-field GPS measurements to further evaluate *DeepGPS*. Specifically, we will investigate the performance of each model component and the impacts of some important parameters.

Visualization. We collect raw GPS measurements using modern smartphones in street canyons of *Region N*, and visualize partial GPS data in Fig. 14. Compared to the ground truth positions as shown in Fig. 14a, GPS estimations from smartphones are quite noisy and deviate from the ground truths greatly. In contrary, *DeepGPS*can effectively correct these erroneous GPS estimations, as shown in

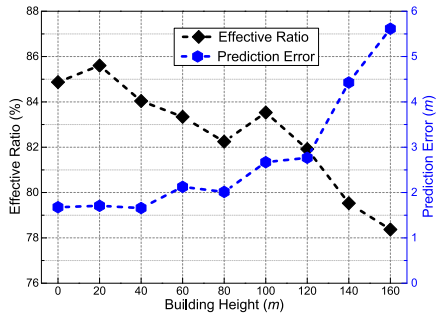

Fig. 12. Effect of building heights on effective ratio and prediction error.

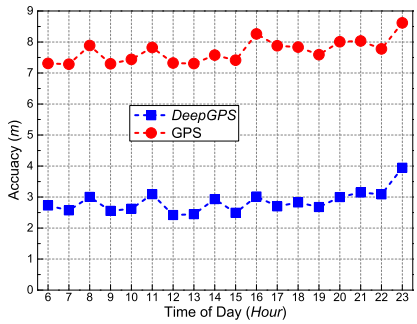

Fig. 13. Effect of time on the accuracy of GPS and *DeepGPS*.

Fig. 14c. In addition to comparison on discrete positions, Fig. 3 compares three complete traces, which are the ground truth trace, smartphone trace, and *DeepGPS*'s corrected trace. Fig. 3 shows that *DeepGPS*improves GPS performance by largely reducing the error from 12.3 meters to 5.2 meters.

Impact of Different Model Components. To understand how input data sources and functional components affect the performance of *DeepGPS*, we conduct various ablation experiments. We treat GPS performance as the baseline, and take variant system designs for comparison. In the following experiments, we remove each component and train the remaining model. All variant models are trained and tested with the same training and testing datasets, respectively. The results are shown in Fig. 16, where "*w.o.*" is short for *without* and "*dist. dec.*" stands for *distance decoder*.

In general, we find that these input data, i.e., building heights, road information, and timestamp, have much larger impacts on *DeepGPS*'s performance than other modules, e.g., the constraint mask and mobility model. If we omit the input of building or road information, *DeepGPS*'s performance is severely degraded to 6.9 meters, with accuracy similar as the GPS error and effective ratio smaller than 60%. Thus, it proves that environment is an important factor that affects GPS positioning performance in urban canyons.

From Fig. 16, we surprisingly find that time has the largest influence on the positioning performance. Without inputting the timestamp matrix, the accuracy of *DeepGPS*will decrease to 7.5 meters that is even larger the GPS error (i.e., 7.2 meters), while the effective ratio is as low as 39.3%. This phenomenon could be explained as follows. In urban canyons, multipath and NLOS satellites are the major causes for GPS performance degradation. For the GPS receiver at a specific place, whether the received GPS signal is reflected or not is mainly determined by the surrounding buildings and the distribution of satellites, i.e., satellite

(a) Smartphone (b) Ground Truth (c) Corrected

Fig. 14. Visualization of partial real-field GPS measurements: (a) Ground truth positions; (b) GPS estimations from smartphones; (c) Corrected positions from *DeepGPS*.

geometry the receiver can observe. Considering that GPS satellites are operating periodically, as a result, when the environment around a receiver remains unchanged, multipath and NLOS satellites at that place have an intrinsic repeatability characteristic due to the approximate repetition of satellite geometry [19]. Fig. 16 shows that skyplot can affect *DeepGPS*'s performance as well. Compared to other input data sources, however, its impact is limited, with accuracy and effective ratio dropping to 4.4 meters and 73.4%, respectively. Compared to the skyplot, timestamp serves as a more crucial factor that can capture the important relationship between satellite geometry and positioning performance.

In our deep neural network model, we use the distance decoder and position decoder to train the encoder together and the results in Fig. 16 also demonstrate the effectiveness of distance decoder, which improves the accuracy from 4.3 meters to 3.6 meters. It confirms that distance decoder can affect position decoder by adding implicit constraints on the position predictions.

We have proposed two functional masks, i.e., constraint mask and mobility mask, to further improve *DeepGPS*. The constraint mask exploits environment information to do the job like map matching and reduces positioning error by filtering out impossible cells, with accuracy improvement by 10%. In addition, the mobility mask further improves *DeepGPS*'s accuracy by about 5%.

Impact of k and ρ. The mobility model uses the latest k correct positions to compute user's moving speed, and assigns a confidence ρ to unreachable cells. We thus perform experiments to study the impact of k and ρ on the positioning accuracy. As shown in Fig. 17, when we utilize more correct positions, i.e., increasing k, for the speed calculation, the accuracy generally increases for a given ρ. However, when $k \geq 5$, more positioning samples bring negligible accuracy improvement.

With the increase of confidence ρ, the accuracy increases for the settings of $k \leq 5$; for other k values, the positioning accuracy increases when $\rho \leq 0.4$, but becomes a bit worse with larger ρ. When the speed calculation is not sufficiently accurate (e.g., with $k \leq 5$ samples), *DeepGPS*tends to equally treat all candidate cells by preferring a larger ρ for the better positioning accuracy. When we can accurately estimate the moving speed (i.e., with $k \geq 5$ samples), the user's actual position should be very likely in these cells covered by the reachable area (i.e., green cells with confidence 1 in Fig. 8) and thus we need a small ρ to filter out the unreachable cells. However, too smaller ρ value (e.g., 0.1) implies that *DeepGPS*will blindly believe in the mobility model and may mistakenly filter out the correct cells for some positioning instances that occasionally fall in the unreachable cells. Thus, too smaller ρ is a bit aggressive and would harm the positioning accuracy. On the other hand, larger ρ value will weaken the filtering capability of the mobility mask, and mistakenly select some unreachable cell as the output, which instead reduces the positioning accuracy. From Fig. 17, we conclude that $k = 5$ and $\rho = 0.4$ are good settings to achieve the best accuracy while avoiding extra computations.

Processing Time. We evaluate the efficiency of *DeepGPS*, and present average processing time of four key modules in Table 5. To process each request, the input representation module takes the most time, e.g., 34.1 *ms*, to construct the three matrices, while the processing time of encoder or any decoder is very little, e.g., < 4 *ms*. In addition, it will take several milliseconds to derive the correct position by searching the element with the maximum probability in the final

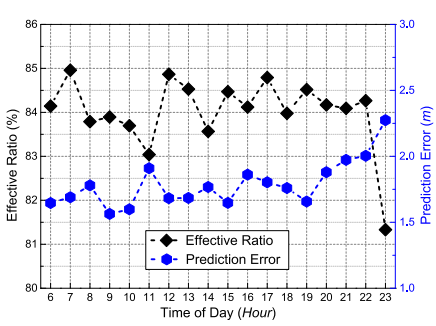

Fig. 15. Effect of time on effective ratio and prediction error.

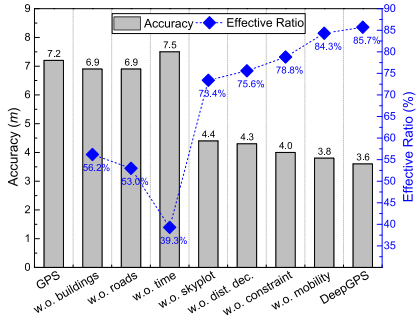

Fig. 16. Impact of each input data source and functional component on performance.

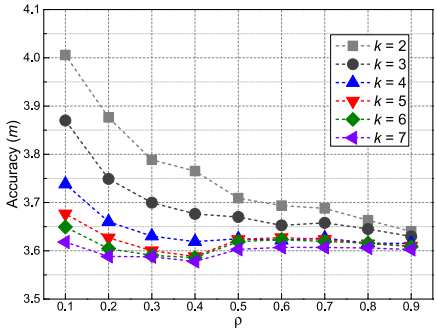

Fig. 17. Impact of various k and ρ settings on the position prediction accuracy.

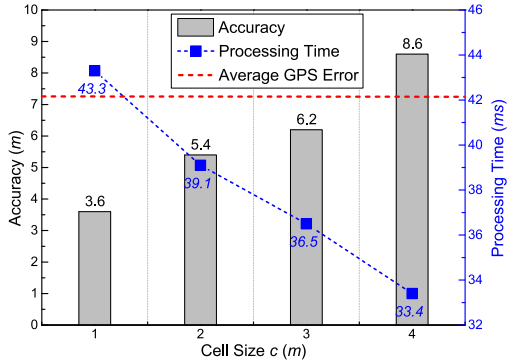

Fig. 18. Impact of cell sizes on the accuracy and processing time.

output matrix. The overall processing time of *DeepGPS*to correct a GPS estimation is about 43 *ms*. As a clear comparison, the state-of-the-art method *Gnome* [39], which recomputes the pseudorange based on 3D building models and satellite positions for position correction, needs several seconds to correct a GPS estimation (excluding the time for offline pre-computations). Hence, training a deep learning model for inferring correct positions is efficient.

Impact of Cell Size. Fig. 18 shows the positioning accuracy and end-to-end processing time under various settings of cell size c. In general, larger cells can reduce the size of input matrices, and thus reduce the whole processing time. However, they bring larger positioning errors at the same time. For example, $4\,m \times 4\,m$ cells lead to positioning error as 8.6 meters, which is larger than the GPS error. In contrary, $1\,m \times 1\,m$ cells improve the accuracy to 3.6 meters, at the cost of 10 *ms* increasing on latency merely. Since the increased processing time is quite small, *DeepGPS*thus adopts $1\,m \times 1\,m$ cells for better positioning performance.

6 Related Work

Tremendous efforts have been devoted to improve GPS accuracy in the urban areas. Vehicles can combine various techniques, e.g., map matching [47], [53] and dead-reckoning [31], with GPS estimations to map their positions to road segments. Pedestrian users can exploit cellular/WiFi signals [24], [30], inertial sensors [15], [62], magnetic compasses [55], and barometer [26], on their smartphones to enhance localization. Other techniques, like cooperative GPS [17] and differential GPS [25], have been proposed to improve GPS accuracy by sharing localization information among multiple receivers. For example, Chen et al. [17] present *BikeGPS* that realize accurate localization of shared bikes in urban canyons by sharing GPS receptions among a group of bikes. These works, however, require extra sensor or cooperation among multiple GPS receivers. Moreover, they do not attack the root cause of GPS error in urban canyons, i.e., NLOS satellite signals.

TABLE 5
Processing Time (in *ms*) of Each Module in *DeepGPS*

Module	Input	Encoder	Position Decoder	Distance Decoder
Time	34.1	3.9	1.6	0.9

Previous works indeed have explored the mitigation of NLOS receptions, which can be classified into three categories, namely *ray-tracing based methods* [21], [67], *shadow matching based methods* [20], [23], [49], [50], [60], [63], [68] and *satellite signal path based methods* [29], [39], [45], [48]. Researches of the first category [21], [67] apply ray-tracing algorithms on satellite signals to correct the pseudorange errors. For example, Zhang et al. [67] propose a 3D mapping database aided GNSS based collaborative positioning method, which exploits the ray-tracing algorithm to correct the NLOS pseudorange for each GPS receiver and employs the factor graph optimization technique to collaboratively optimize the positioning among multiple receivers. This method, however, relies on the 3D building model and collaboration among multiple GPS receivers. For high-quality ray-tracing, some methods require to reconstruct reflective surfaces of street buildings using specialized hardware, e.g., panoramic cameras [54] or LiDAR [57].

In Section 2.3, we have discussed shadow matching based methods and satellite signal path based methods, which leverage proprietary 3D city models and real-time satellite information to explicitly correct GPS estimations. As a representative work of the former category, Ng et al. [49] implement shadow matching for smartphones with a machine learning classifier to distinguish LOS and NLOS satellites. However, shadow matching based methods primarily rely on precise 3D city models, and removing NLOS satellites may reduce the number of usable satellite readings and fail to calculate the receiver's position. In addition, *Gnome* [39], a recent work of the latter category, uses panoramic images from Google Street View to adjust building heights of 3D city model, and then estimates true position from candidate grids by leveraging these building data. *Gnome* heavily relies on third-party resources, which are not widely available, and will incur huge computation overheads. As a result, the restricted availability of 3D building models will largely limit the practical adoptions of these methods.

With the wide availability of GPS trajectory data [70], many works attempt to measure and calibrate GPS errors from the aspect of statistic [28], [42], [46], [51], [58]. For example, Ma et al. [42] assess GPS environment friendliness of urban road segments using historical bus GPS trajectory data. Wu et al. [58] assume GPS errors follow a Gaussian distribution, and locate one single GPS position to a road segment based on a statistical model specially learned from GPS data of that road segment. Different from these works, we exploit massive GPS samples to train a deep neural

TABLE 6
Accuracy Comparison With Other Methods

Method	Ref. [67]	Ref. [49]	*Gnome* [39]	*DeepGPS*
Accuracy (*m*)	7.8	6.0	6.2	3.6

network, which transforms GPS estimations to their correct positions and can serve for a large urban area.

7 Discussion

In this section, we discuss some issues covering *DeepGPS*'s performance comparison, implementation, updating, and data release.

Accuracy Comparisons With Other Works. Previous works heavily rely on precise proprietary 3D building models that are not easily accessible, as a result, we cannot implement existing methods for direct performance comparisons. Instead, we summarize their average positioning accuracy according to their experiment results, and compare *DeepGPS*with three representative works [39], [49], [67] mentioned in previous section on the performance metric of positioning accuracy. As shown in Table 6, we see that *DeepGPS*significantly outperforms existing methods by achieving much better positioning accuracy, e.g., improving the accuracy by 53.8%, 40.0%, and 41.9%, respectively.

Deployment on Smartphones. We have to address two key challenges when deploying *DeepGPS*on the smartphones. First, the execution of *DeepGPS*depends on a well-trained model whose size is a bit large, i.e., about 170 MB, and other resources, e.g., the road network and building survey data, which are usually of large sizes. For example, the road network file of Shenzhen city is about 344 MB, and the building survey data is about 420 MB. Therefore, the deployment of *DeepGPS*consumes at least 934 MB memory, which is a relatively huge storage overhead for the ordinary smartphones. Second, *DeepGPS*will incur considerable computation cost. Given a position fixing request, the system needs to query road network and building survey data to retrieve environment information around the initial GPS position, and then calculate the environment matrix, skyplot matrix and timestamp matrix. Then, it feeds the three matrices into the model to infer positioning error and correct position. Currently, we leave these computations to the powerful server rather than the smartphones.

In the future, we will study how to compress the deep neural network model while retaining its performance by investigating some advanced model compression techniques [38]. Besides, we find that GPS actually performs well in most areas except those areas with densely distributed high buildings. Therefore, we could conduct a survey about the GPS error distribution across a city, and determine the regions where GPS performs poorly. For each region, we prepare an environment package, which only contains the road network and building survey data of that region. Compared to the city-scale road network and building survey data, such region-level environment packages will be much smaller and are friendly to the smartphones. The users can thus download the packages they just need. However, more efforts are required to reduce the computation overheads.

System Resilience and Updating. Even if the urban environment changes, e.g., the construction of new buildings and/or the demolition of old buildings, *DeepGPS*can be still effective. It is because our deep neural network model captures the general mapping relationship between GPS estimations and positioning contexts. Once the urban environment changes are recorded in the road network file or building survey data, such information can be immediately encoded in the environment matrix. In addition, we propose the constraint mask $\mathcal{C}_{env}$, which embeds prior knowledge of surrounding environment, e.g., buildings, to constrain the prediction of correct position. As a result, if an area is occupied by new buildings, the cells covered by the buildings are marked as unavailable in the constraint mask $\mathcal{C}_{env}$. With these designs, our system can effectively correct GPS estimations, and thus be resilient to environment changes.

Despite above novel designs, we still suggest to periodically retrain the model using the latest resource data, including the newly collected GPS samples, the latest road network and building survey data. Periodical model retraining aims to timely update the mapping relationship between GPS estimations and positioning contexts. Specifically, after a given period, e.g., three months, we can retrain the model using the latest resource data. In general, the retraining process could be completed within a few hours. For example, it takes about 6.6 hours to train the model for Shenzhen city, China. During the model retraining, the existing model can be still used to serve position fixing requests. Once the model retraining is done, we will replace the old model with the updated one to provide more accurate position fixing service.

Data Release for Reproducibility. We share the source code of *DeepGPS*'s implementation and the real-field GPS samples [8] for the community to reproduce our results and inspire future studies.

8 Conclusion

In this paper, we present a deep learning enhanced GPS positioning system – *DeepGPS*, which leverages an encoder-decoder network model to implicitly map erroneous GPS estimations to ground truth positions. More specifically, *DeepGPS*fuses multiple factors that affect GPS accuracy in urban canyons, and inputs them into the model to predict both positioning error and correct position through two parallel decoders. We further enhance *DeepGPS*with a novel constraint mask design by filtering out inaccessible candidate locations, and enable continuous localization using a simple yet effective mobility model. The system has been implemented and evaluated. Extensive experiments based on a large-scale bus trajectory dataset and real-field GPS measurements show that *DeepGPS*can significantly enhance GPS positioning in urban canyons, e.g., on average effectively correcting 90.1% GPS estimations with accuracy improvement by 64.6%.

References

[1] Celestrak. Accessed: Jul. 2022. [Online]. Available: https://celestrak.com/
[2] GnssStatus. Accessed: Jul. 2022. [Online]. Available: https://developer.android.com/reference/android/location/GnssStatus
[3] GPS errors and biases. Accessed: Jul. 2022. [Online]. Available: http://what-when-how.com/category/gps/

[4] Open Street Map. Accessed: Jul. 2022. [Online]. Available: https://www.openstreetmap.org/

[5] PostGIS. Accessed: Jul. 2022. [Online]. Available: https://postgis.net/

[6] PostgreSQL. Accessed: Jul. 2022. [Online]. Available: https://www.postgresql.org/

[7] Pytorch. Accessed: Jul. 2022. [Online]. Available: https://pytorch.org/

[8] DeepGPS source code. Accessed: Jul. 2022. [Online]. Available: https://github.com/bducgroup/DeepGPS

[9] Wikipedia: List of cities with the most skyscrapers. Accessed: Jul. 2022. [Online]. Available: https://en.wikipedia.org/wiki/List_of_cities_with_the_most_skyscrapers

[10] A. Adams and P. Vamplew, "Encoding and decoding cyclic data," *South Pacific J. Natural Sci.*, vol. 16, pp. 54–58, 1998.

[11] D. C. Agnew and K. M. Larson, "Finding the repeat times of the GPS constellation," *GPS Solutions*, vol. 11, no. 1, pp. 71–76, 2007.

[12] F. Ahmad, H. Qiu, R. Eells, F. Bai, and R. Govindan, "CarMap: Fast 3D feature map updates for automobiles," in *Proc. USENIX Symp. Netw. Syst. Des. Implementation*, 2020, pp. 1063–1081.

[13] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 26–38, Nov. 2017.

[14] R. Ayyalasomayajula et al., "Deep learning based wireless localization for indoor navigation," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–14.

[15] C. Bo, X.-Y. Li, T. Jung, X. Mao, Y. Tao, and L. Yao, "SmartLoc: Push the limit of the inertial sensor based metropolitan localization using smartphone," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2013, pp. 195–198.

[16] J. Bressler, P. Reisdorf, M. Obst, and G. Wanielik, "GNSS positioning in non-line-of-sight context - a survey," in *Proc. IEEE Int. Conf. Intell. Transp. Syst.*, 2016, pp. 1147–1154.

[17] K. Chen and G. Tan, "BikeGPS: Accurate localization of shared bikes in street canyons via low-level GPS cooperation," in *Proc. ACM Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2018, pp. 150–162.

[18] X. Chen, F. Dovis, S. Peng, and Y. Morton, "Comparative studies of GPS multipath mitigation methods performance," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 49, no. 3, pp. 1555–1568, Jul. 2013.

[19] W. Dai, D. Huang, and C. Cai, "Multipath mitigation via component analysis methods for GPS dynamic deformation monitoring," *GPS Solutions*, vol. 18, no. 3, pp. 417–428, 2014.

[20] V. Drevelle and P. Bonnifait, "iGPS: Global positioning in urban canyons with road surface maps," *IEEE Intell. Transp. Syst. Mag.*, vol. 4, no. 3, pp. 6–18, Fall 2012.

[21] R. Ercek, P. De Doncker, and F. Grenez, "NLOS-multipath effects on pseudo-range estimation in urban canyons for GNSS applications," in *Proc. IEEE 1st Eur. Conf. Antennas Propag.*, 2006, pp. 1–6.

[22] J. Feng et al., "DeepMM: Deep learning based map matching with data augmentation," *IEEE Trans. Mobile Comput.*, vol. 21, no. 7, pp. 2372–2384, Jul. 2022.

[23] P. D. Groves, "Shadow matching: A new GNSS positioning technique for urban canyons," *J. Navigation*, vol. 64, no. 3, pp. 417–430, Jul. 2011.

[24] F. Gustafsson and F. Gunnarsson, "Mobile positioning using wireless networks: Possibilities and fundamental limitations based on available wireless network measurements," *IEEE Signal Process. Mag.*, vol. 22, no. 4, pp. 41–53, Jul. 2005.

[25] W. Hedgecock, M. Maroti, J. Sallai, P. Volgyesi, and A. Ledeczi, "High-accuracy differential tracking of low-cost GPS receivers," in *Proc. ACM Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2013, pp. 221–234.

[26] P.-F. Ho, C.-C. Hsu, J.-C. Chen, and T. Zhang, "Using barometer on smartphones to improve GPS navigation altitude accuracy," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2018, pp. 741–743.

[27] L.-T. Hsu, "GNSS multipath detection using a machine learning approach," in *Proc. IEEE Int. Conf. Intell. Transp. Syst.*, 2017, pp. 1–6.

[28] L.-T. Hsu, "Analysis and modeling GPS NLOS effect in highly urbanized area," *GPS Solutions*, vol. 22, no. 1, pp. 1–12, 2018.

[29] L.-T. Hsu, Y. Gu, and S. Kamijo, "3D building model-based pedestrian positioning method using GPS/GLONASS/QZSS and its reliability calculation," *GPS Solutions*, vol. 20, no. 3, pp. 413–428, 2016.

[30] M. Ibrahim et al., "Wi-Go: Accurate and scalable vehicle positioning using WiFi fine timing measurement," in *Proc. ACM Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2020, pp. 312–324.

[31] Y. Jiang et al., "Carloc: Precise positioning of automobiles," in *Proc. ACM Conf. Embedded Netw. Sensor Syst.*, 2015, pp. 253–265.

[32] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 694–711.

[33] E. D. Kaplan and C. Hegarty, *Understanding GPS/GNSS: Principles and Applications*. Norwood, MA, USA: Artech House, 2017.

[34] D. Karamshuk, A. Noulas, S. Scellato, V. Nicosia, and C. Mascolo, "Geo-spotting: Mining online location-based services for optimal retail store placement," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2013, pp. 793–801.

[35] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[36] C.-T. Lin, Y.-Y. Wu, P.-H. Hsu, and S.-H. Lai, "Multimodal structure-consistent image-to-image translation," in *Proc. Conf. Assoc. Advanc. Artif. Intell.*, 2020, pp. 11490–11498.

[37] J. Liu, B. Priyantha, T. Hart, H. S. Ramos, A. A. Loureiro, and Q. Wang, "Energy efficient GPS sensing with cloud offloading," in *Proc. ACM Conf. Embedded Netw. Sensor Syst.*, 2012, pp. 85–98.

[38] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proc. ACM Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2018, pp. 389–400.

[39] X. Liu, S. Nath, and R. Govindan, "Gnome: A practical approach to NLOS mitigation for GPS positioning in smartphones," in *Proc. ACM Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2018, pp. 163–177.

[40] Z. Liu, Z. Gong, J. Li, and K. Wu, "Mobility-aware dynamic taxi ridesharing," in *Proc. IEEE Int. Conf. Data Eng.*, 2020, pp. 961–972.

[41] Z. Liu, Z. Li, K. Wu, and M. Li, "Urban traffic prediction from mobility data using deep learning," *IEEE Netw.*, vol. 32, no. 4, pp. 40–46, Jul./Aug. 2018.

[42] L. Ma et al., "Estimating urban road GPS environment friendliness with bus trajectories: A city-scale approach," *Sensors*, vol. 20, no. 6, 2020, Art. no. 1580.

[43] J. Marshall, "Creating and viewing skyplots," *GPS Solutions*, vol. 6, no. 1, pp. 118–120, 2002.

[44] J.-I. Meguro, T. Murata, J.-I. Takiguchi, Y. Amano, and T. Hashizume, "GPS multipath mitigation for urban area using omnidirectional infrared camera," *IEEE Trans. Intell. Transp. Syst.*, vol. 10, no. 1, pp. 22–30, Mar. 2009.

[45] S. Miura, L.-T. Hsu, F. Chen, and S. Kamijo, "GPS error correction with pseudorange evaluation using three-dimensional maps," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 6, pp. 3104–3115, Dec. 2015.

[46] M. Modsching, R. Kramer, and K. ten Hagen, "Field trial on GPS accuracy in a medium size city: The influence of built-up," in *Proc. 3rd Workshop Positioning, Navigation Commun.*, 2006, pp. 209–218.

[47] P. Newson and J. Krumm, "Hidden Markov map matching through noise and sparseness," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geogr. Inf. Syst.*, 2009, pp. 336–343.

[48] H.-F. Ng, G. Zhang, and L.-T. Hsu, "A computation effective range-based 3D mapping aided GNSS with NLOS correction method," *J. Navigation*, vol. 73, no. 6, pp. 1202–1222, Nov. 2020.

[49] H.-F. Ng, G. Zhang, and L.-T. Hsu, "Robust GNSS shadow matching for smartphones in urban canyons," *IEEE Sensors J.*, vol. 21, no. 16, pp. 18307–18317, Aug. 2021.

[50] S. Peyraud et al., "About non-line-of-sight satellite detection and exclusion in a 3D map-aided localization algorithm," *Sensors*, vol. 13, no. 1, pp. 829–847, 2013.

[51] J. Schipperijn, J. Kerr, S. Duncan, T. Madsen, C. D. Klinker, and J. Troelsen, "Dynamic accuracy of GPS receivers for use in health research: A novel method to assess GPS accuracy in real-world settings," *Front. Public Health*, vol. 2, 2014, Art. no. 21.

[52] K. L. Senior, J. R. Ray, and R. L. Beard, "Characterization of periodic variations in the GPS satellite clocks," *GPS Solutions*, vol. 12, no. 3, pp. 211–225, 2008.

[53] Z. Shen, W. Du, X. Zhao, and J. Zou, "DMM: Fast map matching for cellular data," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2020, pp. 1–14.

[54] S. Tay and J. Marais, "Weighting models for GPS pseudorange observations for land transportation in urban canyons," in *Proc. 6th Eur. Workshop GNSS Signals Signal Process.*, 2013, Art. no. 4p.

[55] C.-C. Wang et al., "MVP: Magnetic vehicular positioning system for GNSS-denied environments," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2021, pp. 531–544.

[56] G. Wang, X. Xie, F. Zhang, Y. Liu, and D. Zhang, "bCharge: Data-driven real-time charging scheduling for large-scale electric bus fleets," in *Proc. IEEE Real-Time Syst. Symp.*, 2018, pp. 45–55.

[57] W. Wen, G. Zhang, and L.-T. Hsu, "Correcting NLOS by 3D LiDAR and building height to improve GNSS single point positioning," *Navigation*, vol. 66, no. 4, pp. 705–718, 2019.
[58] H. Wu, W. Sun, and B. Zheng, "Is only one GPS position sufficient to locate you to the road network accurately?," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2016, pp. 740–751.
[59] H. Xie, T. Gu, X. Tao, H. Ye, and J. Lu, "A reliability-augmented particle filter for magnetic fingerprinting based indoor localization on smartphone," *IEEE Trans. Mobile Comput.*, vol. 15, no. 8, pp. 1877–1892, Aug. 2016.
[60] H. Xu, A. Angrisano, S. Gaglione, and L.-T. Hsu, "Machine learning based LOS/NLOS classifier and robust estimator for GNSS shadow matching," *Satell. Navigation*, vol. 1, no. 1, pp. 1–12, 2020.
[61] Y. Yang, J. Jiang, and M. Su, "Comparison of satellite repeat shift time for GPS, BDS, and Galileo navigation systems by three methods," *Algorithms*, vol. 12, no. 11, 2019, Art. no. 233.
[62] Z. Yang, C. Wu, Z. Zhou, X. Zhang, X. Wang, and Y. Liu, "Mobility increases localizability: A survey on wireless indoor localization using inertial sensors," *ACM Comput. Surv.s*, vol. 47, no. 3, pp. 1–34, 2015.
[63] R. Yozevitch, B. Ben-Moshe, and A. Dvir, "GNSS accuracy improvement using rapid shadow transitions," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 3, pp. 1113–1122, Jun. 2014.
[64] P. A. Zandbergen, "Positional accuracy of spatial data: Non-normal distributions and a critique of the national standard for spatial data accuracy," *Trans. GIS*, vol. 12, no. 1, pp. 103–130, 2008.
[65] K. C. Zeng et al., "All your GPS are belong to us: Towards stealthy manipulation of road navigation systems," in *Proc. USENIX Secur. Symp.*, 2018, pp. 1527–1544.
[66] D. Zhang, J. Huang, Y. Li, F. Zhang, C. Xu, and T. He, "Exploring human mobility with multi-source data at extremely large metropolitan scales," in *Proc. ACM Annu. Int. Conf. Mobile Comput. Netw.*, 2014, pp. 201–212.
[67] G. Zhang, H.-F. Ng, W. Wen, and L.-T. Hsu, "3D mapping database aided GNSS based collaborative positioning using factor graph optimization," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 10, pp. 6175–6187, Oct. 2021.
[68] G. Zhang, W. Wen, B. Xu, and L.-T. Hsu, "Extending shadow matching to tightly-coupled GNSS/INS integration system," *IEEE Trans. Veh. Technol*, vol. 69, no. 5, pp. 4979–4991, May 2020.
[69] Y. Zhang et al., "Route prediction for instant delivery," *Proc. ACM Interactive, Mobile, Wearable Ubiquitous Technol.*, vol. 3, no. 3, pp. 1–25, 2019.
[70] Y. Zheng, "Trajectory data mining: An overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, pp. 1–41, 2015.

Zhidan Liu (Member, IEEE) received the PhD degree in computer science and technology from Zhejiang University, Hangzhou, China, in 2014. After that, he worked as a research fellow with Nanyang Technological University, Singapore. He is currently an Associate Professor with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China. His research interests include mobile computing, big data analytics, Internet of Things, and urban computing. He is a member ACM, and CCF.

Jiancong Liu is currently working toward the fourth-year undergraduation degree with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China, under the supervision of Dr. Zhidan Liu. His research interests are in the areas of trajectory data analysis and mobile computing.

Xiaowen Xu is currently working toward the third-year undergraduation degree with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen, China, under the supervision of Dr. Zhidan Liu. Her research interests are in the areas of trajectory data analysis and mobile computing.

Kaishun Wu (Member, IEEE) received the PhD degree in computer science and engineering from The Hong Kong University of Science and Technology (HKUST), Hong Kong, China, in 2011. After that, he worked as a research Assistant Professor with HKUST. In 2013, he joined Shenzhen University as a distinguish professor. Currently, he is a professor with the DSA & IoT Thrust Area under the Information Hub, Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China. He has coauthored 2 books and published more than 100 high quality research papers in international leading journals and primer conferences, like *IEEE Transactions on Mobile Computing*, *IEEE Transactions on Parallel and Distributed Systems*, ACM MobiCom, IEEE INFOCOM. He is the inventor of 6 US and more than 90 Chinese pending patents. He received 2012 Hong Kong Young Scientist Award, 2014 Hong Kong ICT Awards: Best Innovation, and 2014 IEEE ComSoc Asia-Pacific Outstanding Young Researcher Award. He is an IET fellow.