Glow in the Dark: Smartphone Inertial Odometry for Vehicle Tracking in GPS Blocked Environments

Ruipeng Gao, Xuan Xiao, Shuli Zhu, Weiwei Xing, Chi Li, Lei Liu, Li Ma, and Hua Chai

***Abstract*—Although vehicle location-based services are prevalent outdoors, we are back into darkness in many GPS blocked environments, such as tunnels, indoor parking garages, and multilevel flyovers. Existing smartphone-based solutions usually adopt inertial dead reckoning to infer the trajectory, but low-quality inertial sensors in phones are plagued by heavy noises, causing unbounded localization errors through double integrations for movements. In this article, we propose *VeTorch*, a smartphone inertial odometry that devises an inertial sequence learning framework to track vehicles in real time when GPS signal is not available. Specifically, we transform the inertial dynamics from the phone to the vehicle regardless of the arbitrary phone's placement in the car and explore a temporal convolutional network to learn the vehicle's moving dependencies directly from the inertial data. To tackle the heterogeneous smartphone properties and driving habits, we propose a federated learning-based active model training mechanism to produce customized models for individual smartphones, without incurring user privacy issues. We implement a highly efficient prototype and conduct extensive experiments on two large-scale real-world traffic data sets collected by a modern ride-hailing platform. Our results outperform the state-of-the-art vehicular inertial dead-reckoning solutions on both accuracy and efficiency.**

***Index Terms*—Federated learning, inertial sequence learning, location-based services, vehicle tracking.**

I. Introduction

THANKS to the explosion of GPS systems and devices, drivers can easily track their vehicles in most urban areas. Such location information is pivotal to many vehicular location-based services, including route planning, vehicle dispatching, and self-driving. However, whenever we drive into GPS blocked areas, such as tunnels, underground parking garages, or multilevel flyovers, we lose the location awareness. Not only do we have to endure the location "blindness," frequently we get confused on each road junction we drive through.

Enabling vehicular tracking services in GPS blocked environments will support many practical driving demands, including generating the right senses of control and inducing calmness psychologically, both significantly enhancing the driving experience. For example, when driving from the Boston airport to the MIT Museum, there are four tunnels lasting for 2.7 miles with six inside road junctions along the path, and drivers have to identify each correct road junction and take immediate actions (e.g., lane shifting).

However, tracking vehicle without GPS signals is far from straightforward. Although vehicles display instant speed values at the dashboard, we can hardly exploit such data except deploying a dedicated onboard diagnostics (OBD) device to car's busline, which is not always suitable for the crowd. Mainstream indoor localization technologies leverage the radio frequency (RF) signals, such as Wi-Fi [1], [2] and Bluetooth [3], which can hardly penetrate in many large-scale enclosed environments, such as tunnels and garages. Deploying sensor networks is always time consuming and cost expensive. In addition, the lack of network infrastructure (e.g., cellular towers) also indicates no cloud service, thus both data storage and computation occur locally at the device, which impedes the application of deep learning methods [4], [5] for inertial tracking in real time.

In this article, we propose *VeTorch* which only leverages the smartphone's inertial data to infer vehicle's locations in real time. It does not require any additional sensors and is scalable for fast and customized deployment on individual smartphones without incurring user privacy issues.

Such an inertial and phone-only approach entails a series of nontrival challenges. First, the inertial data is collected via smartphones, thus we need to carefully transform such dynamics from the phone to the vehicle despite arbitrary phone's placements in the car. Second, low-quality inertial measurement unit (IMU) sensors in commodity phones are plagued by heavy noises, causing unbounded localization errors through double integrations for movements. Finally, the heterogeneous smartphone hardware properties and user driving habits make it challenging to achieve optimal performance with a unified model trained on an existing data set. Efficient training of customized models without incurring privacy issues is critical for practical use.

Manuscript received June 16, 2020; revised November 16, 2020 and January 23, 2021; accepted February 28, 2021. Date of publication March 8, 2021; date of current version August 6, 2021. This work was supported in part by NSFC under Grant 62072029, Grant 61872027, and Grant 61876018; in part by the National Key Research and Development Program of China under Grant 2018YFB1601000; in part by Beijing NSF under Grant L192004; in part by the Open Research Fund of the State Key Laboratory of Integrated Services Networks under Grant ISN21-16; in part by the DiDi Research Collaboration Plan; and in part by the CCF-Tencent Open Fund. *(Corresponding author: Ruipeng Gao.)*

Ruipeng Gao is with the School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China, and also with the State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an 710071, China (e-mail: rpgao@bjtu.edu.cn).

Xuan Xiao, Shuli Zhu, and Weiwei Xing are with the School of Software Engineering, Beijing Jiaotong University, Beijing 100044, China (e-mail: xiaoxuan@bjtu.edu.cn; zhushuli@bjtu.edu.cn; wwxing@bjtu.edu.cn).

Chi Li, Lei Liu, Li Ma, and Hua Chai are with the Maps and Public Transportation Department, DiDi Corporation, Beijing 100193, China (e-mail: lichi@didiglobal.com; liuleifrey@didiglobal.com; malimarey@didiglobal.com; chaihua@didiglobal.com).

Digital Object Identifier 10.1109/JIOT.2021.3064342

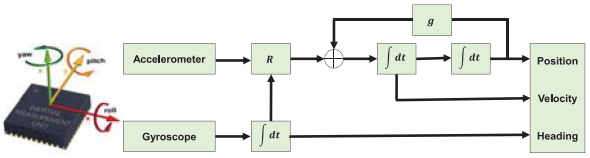


Fig. 1. Diagram of a strapdown inertial navigation system. R is the acceleration projection, and g denotes the gravity update.

Our solution consists of several components to deal with the above challenges, aiming to infer accurate and real-time vehicular locations regardless of GPS signals. Specifically, we make the following contributions.

1) We effectively transform inertial dynamics from the phone to the vehicle, thus eliminate impacts of phone's placement in car and reduce inertial noises.
2) We devise a TCN architecture to yield vehicle's movements directly from inertial sequences, instead of continuous integrations. We further enhance the robustness by fusing with auxiliary attributes, such as historical trajectories, speed trend, and social information.
3) We explore a federated learning solution to produce customized models for fast deployment on individual smartphones, without the need of raw driving data uploading, eliminating the privacy issues.
4) We develop a highly efficient prototype and conduct extensive experiments on two large-scale real-world data sets collected by a modern ride-hailing platform. Results have shown our effectiveness compared with the state of the art.

II. Preliminary

When tracking vehicles in GPS blocked environments, current ride-hailing platforms always build a strapdown inertial navigation system as the odometry for vehicle dead reckoning (Fig. 1). Especially, the odometry exploits smartphone's inertial data as inputs, i.e., the 3-axis accelerations by accelerometer and the 3-axis angular rate by the gyroscope, both measured at the phone's coordinate system.

Intuitively, instead of directly double integrating accelerations into distance ($\boldsymbol{x_t} = \int\int \boldsymbol{a_t} dtdt$), they leverage vehicle's heading direction $\boldsymbol{H_t}$ (unit vector) and speed amplitude s_t to infer the location: $\boldsymbol{x_t} = \int \boldsymbol{H_t} \cdot s_t dt$, where s_t is calculated as $s_t = \int a_t dt$, integration of the acceleration amplitude along vehicle's heading direction. Although there are still two integrations, the heading direction can be measured reliably by gyroscope.

In practice, accelerations are always denoised by a first-order low-pass filter. Since the phone's coordinate system is not aligned with the vehicle, estimating vehicle's absolute heading direction is not trivial. They have explored an extended Kalman filter (EKF) solution [6] to track vehicle's attitude, i.e., $\mathbf{q} = [q_1, q_2, q_3, q_4]^T$ in the quaternion expression.

Step 1)—Attitude Prediction: They use gyroscope readings to predict the vehicle's attitude in the next state

$$\hat{\boldsymbol{q}}_{t+1} = \begin{bmatrix} \frac{\boldsymbol{\omega}}{|\boldsymbol{\omega}|} \sin\left(\frac{|\boldsymbol{\omega}|}{2}\Delta t\right) \\ \cos\left(\frac{|\boldsymbol{\omega}|}{2}\Delta t\right) \end{bmatrix} \otimes \boldsymbol{q}_t \tag{1}$$

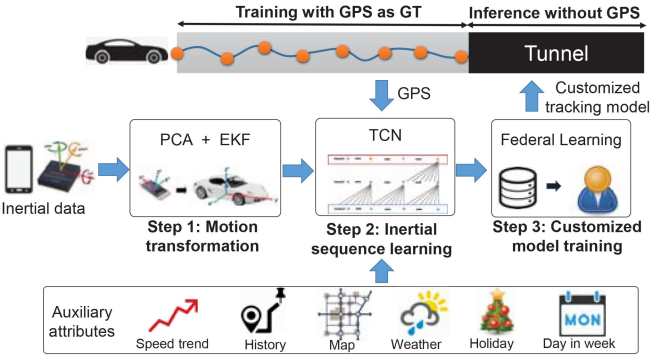


Fig. 2. *VeTorch* consists of three steps: motion transformation from phones to vehicles, inertial sequence learning to model vehicle's movements, and customized location inference for deployment on individual phones. We also leverage auxiliary attributes as bonus features.

where $\boldsymbol{\omega}$ denotes the current angular rate from gyroscope. They also calculate the state covariance matrix $P_{t+1|t}$ as

$$P_{t+1|t} = \Phi P_{t|t} \Phi^T + Q \tag{2}$$

where Φ is the state transition matrix, and Q is the noise variance matrix.

Step 2)—Attitude Updating: They leverage accelerometer readings to update the attitude prediction. First, the Kalman gain K is computed as

$$K = P_{t+1|t} H^T \left(H P_{t+1|t} H^T + R\right)^{-1} \tag{3}$$

where H is the observation matrix from acceleration $\boldsymbol{a}$, and R is the measurement noise. They then update the corresponding covariance matrix

$$P_{t+1|t+1} = (I - KH) P_{t+1|t} (I - KH)^T + KRK^T \tag{4}$$

where I presents the identity matrix. Finally, they update vehicle's attitude based on the Kalman gain

$$\boldsymbol{q}_{t+1} = \hat{\boldsymbol{q}}_{t+1} + K\left(\boldsymbol{a} - H\hat{\boldsymbol{q}}_{t+1}\right). \tag{5}$$

Given the attitude, smartphone's accelerations are projected to vehicle's heading direction as its forwarding accelerations. However, due to the low-quality inertial sensors in commodity smartphones, large drifts and noises appear frequently and are easily accumulated to extreme location errors.

III. Design Overview

In this article, we propose *VeTorch* which utilizes only smartphone's inertial data to track vehicle's location in GPS block environments. Fig. 2 depicts our system architecture which is composed of the training and inference process. First, when vehicles are driven outdoors, we collect their GPS locations as the ground truth to train the tracking model, which transforms inertial sequences to trajectories. Next, when vehicles enter GPS blocked environments such as tunnels, they exploit the pretrained smartphone-specific model to infer locations in real time. In a word, GPS readings are only used to train the model, but not evolved in location inference.

VeTorch leverages a 3-step process to train the customized tracking model. Step 1) *motion transformation* (Section IV),

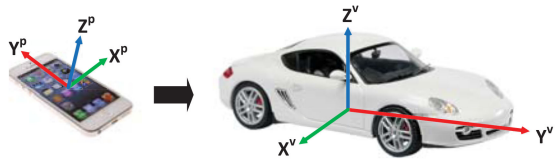


Fig. 3. Data transformation from the smartphone (X^p, Y^p, Z^p) to the vehicle (X^v, Y^v, Z^v).

we use principal component analysis (PCA) and EKF to transform phone's inertial dynamics to the vehicle, regardless of arbitrary phone's placement in car. Step 2) *inertial sequence learning* (Section V), we explore a sequence-to-sequence model to infer vehicle's movements directly from inertial observations, instead of conducting continuous integrations. Step 3) *customized model training* (Section VI), we devise a federated learning mechanism to train customized models for individual smartphones with a fast, secure, and active deployment.

Besides using basic inertial readings as inputs, *VeTorch* can also leverage more *auxiliary attributes* (Section VII) as bonus features to improve the tracking accuracy. For example, we explore a bidirectional recurrent neural network to recover historical trajectories in GPS blocked environments and produce historical speed features at fine-grained areas. We also predict future speed trend based on recent GPS trajectory before losing the signal reception. Moreover, we further implement a Web crawler to extract social attributes as additional features, including the road map, weather condition, holiday index, and the day in week information.

IV. Motion Transformation

The smartphone inertial data, i.e., 3-axis accelerations (a_X^p, a_Y^p, and a_Z^p) and 3-axis angular rates (ω_X^p, ω_Y^p, and ω_Z^p) in Fig. 3, are measured in individual phone's coordinate system, thus they are not aligned within the same reference system due to arbitrary placements of smartphones in car. In this section, we transform such motion dynamics from the smartphone to corresponding movements of the vehicle, which are further learned via our TCN model.

In addition, since the inertial data is highly noisy and susceptible to various disturbances from vehicle's vibration and road conditions, we only produce the vehicle's forwarding acceleration a_Y^v and heading direction ω_Z^v as inputs for location inference, i.e., $\boldsymbol{\Delta v} = f(a_Y^v, \omega_Z^v)$, where $\boldsymbol{v}$ represents vehicle's speed on the ground. This tracking model simplifies the spatial tracking problem onto the 2-D horizontal plane, thus eliminates impacts of inertial noises along the other two axes (i.e., a_X^v, a_Z^v, ω_X^v, and ω_Y^v).

In order to transform accelerations from the smartphone to the vehicle, we describe how to estimate the phone's placement in car, i.e., estimating vehicle's axes (X^v, Y^v, Z^v) in the phone's coordinate system (X^p, Y^p, Z^p) (Fig. 3). First, when the car is static, the gravity direction corresponds to the Z^v-axis. Second, we deduct accelerations onto the horizontal plane and explore a PCA [7] algorithm to yield the forward direction Y^v. The intuition is that vehicle's accelerate/decelerate

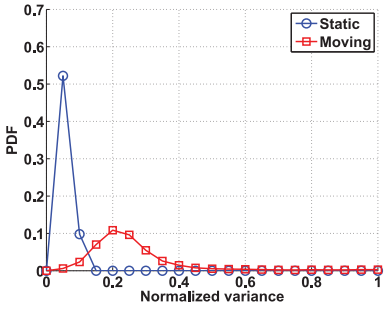


Fig. 4. Distribution of normalized standard deviation during static and moving states.

operations indicate its forward direction with the maximum accelerations. Finally, the rest X^v is calculated as the cross product of Z^v and Y^v. Below, we explain the algorithm in detail.

Step 1 (Z^v-Axis Estimation): The Z^v-axis and the gravity are on exact opposite directions when the vehicle is on level ground. There are various techniques [8] to extract the gravity components at reasonable accuracy (e.g., iOS has an API to obtain the gravity vector). Since vehicle's movements will cause small and random disturbances, we use a low-pass Butterworth filter for denoising and average the extracted gravity samples within a time window. Given the Z^v-axis, the vehicle's rotation for a turn on the xy-plane can be derived reliably by the gyroscope, thus we can identify whether the vehicle is moving along straight lines or making a turn.

Step 2 (Moving Detection): To estimate the vehicle's forwarding direction, we need to detect whether the vehicle is moving. To this end, we calculate the variance of accelerations on three axes averaged within a sliding window. In practice, due to inevitable vibrations, small accelerations always exist and form the variance, thus we consider the vehicle static if the variance is smaller than a threshold. Fig. 4 shows the normalized variance distribution during static and moving states. We can see that the overlapping portion is small compared to the total areas.

Step 3 (Y^v-Axis Estimation): Theoretically, vehicle's accelerations during straight driving are nonzero on the Y^v-axis only (assuming gravity is removed). In practice, there are some noises on the X^v-axis due to inevitable vibrations. Suppose that the direction of acceleration on the xy-plane is represented by a unit vector $(\cos\theta, \sin\theta)$ on the xy-plane of the vehicle. Given a sequence of accelerations, we first project them onto the xy-plane via the gravity direction. Denote the projected results (x_i, y_i), $i = 1, \ldots, k$. We cast the forward direction estimation problem as a PCA approach by deriving the minimum acceleration deviations along the true Y^v-axis, i.e.,

$$\min_{\theta} \sum_{i=1}^{k} (x_i \cos\theta + y_i \cos\theta)^2 = aF \sin 2\theta + b \cos 2\theta + c = \sqrt{a^2 + b^2} \sin(2\theta + \gamma) + c \tag{6}$$

where $a = \sum_{i=1}^{k} x_i y_i$, $b = \sum_{i=1}^{k} ([x_i^2 - y_i^2]/2)$, $c = \sum_{i=1}^{k} ([x_i^2 + y_i^2]/2)$, and $\tan\gamma = (b/a)$. This equation has

a closed-form solution

$$\theta_1 = \frac{1}{4}\pi - \frac{\gamma}{2}, \theta_2 = \frac{5}{4}\pi - \frac{\gamma}{2}. \tag{7}$$

We observe that there are two optimal solutions to θ, representing the forward and backward directions of the vehicle, denoted by θ_1 and θ_2. Next, we decide which is the correct forward direction Y^v.

Step 4 (Forward/Backward Estimation): First, we use θ_1 to compute the corresponding (X^v, Y^v, Z^v) axes in the phone's coordinate system, and project accelerations to the vehicle. When the vehicle starts moving from static for a short time, there should be more positive accelerations on its forward direction. We use a sliding time window and calculate the proportion of positive accelerations as

$$\rho = \frac{1}{k}\sum_{i=1}^{k} y_i^3. \tag{8}$$

Note that we use the cubic form rather than a linear form to better reduce small noises caused by the vibration of the vehicle. Then, we put this value into a sigmoid function and derive a probability of how likely θ_1 matches θ

$$p(\theta = \theta_1) = \frac{1}{1 + e^{-\rho}}. \tag{9}$$

V. Inertial Sequence Learning

After the motion transformation, *VeTorch* takes a sequence as the input and outputs a corresponding sequence at each timestamp, e.g., directly transforming accelerations to the velocity deviation. The goal of inertial sequence learning is to find a network $f : x_{1:t} \mapsto y_{1:t}$ such that minimizes the differences between actual observations and corresponding predictions.

A. Temporal Convolutional Network

Recurrent neural networks (RNNs) (e.g., LSTM [9] and GRU [10]) have become the most popular architecture for sequence modeling, while they are usually computation intensive due to the large number of parameters, making them inefficient on mobile devices. Instead, we choose the TCN [11], which is also effective on certain tasks with long-range contexts, e.g., audio synthesis [12] and machine translation [13], while is more suitable for deployment on smartphones.

There are two basic components in the TCN framework for sequence learning, i.e., the *causal convolutions* and the *dilated convolutions*. Below, we explain them in detail.

1) Causal Convolutions: In order to ensure no leakage from the future into the past, TCN leverages causal convolutions where the output at the current time is only convolved with the elements from the current and earlier time. For 1-D inertial sequence, the causal convolution operator is formalized as

$$\text{CausalConv}(x_t) = x_t * F(k) = \sum_{i=0}^{k-1} x_{t-i} \cdot F_i \tag{10}$$

where k denotes the kernel size, and F_i is the parameter matrix of the convolutional filter.

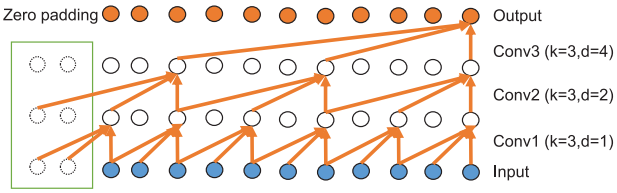


Fig. 5. Causal dilated convolutions.

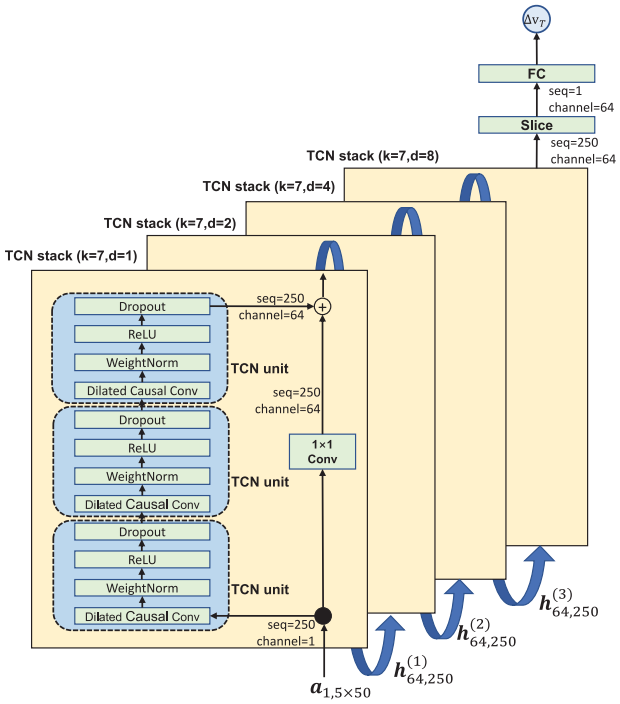


Fig. 6. Model architecture. Our network consists of four TCN stacks, one slice layer, and one FC layer.

2) Dilated Convolutions: In order to derive an exponentially large receptive field along the long history, TCN conducts dilated convolutions on the 1-D vectors. Formally, when combined with causal convolutions, the dilated convolution operator is defined as

$$\text{DilatedConv}(x_t) = x_t * D(k, d) = \sum_{i=0}^{k-1} x_{t-i\cdot d} \cdot D_i \tag{11}$$

where k is the filter size, d is the dilation factor, and $D(k, d)$ denotes the convolutional filter. We summate k multiplied results from current time t to $t - (k-1)\cdot d$, thus significantly increases the receptive field (Fig. 5). In addition, a dilated convolution reduces to a regular convolution when $d = 1$.

B. Model Architecture

Our model exploits the inertial sequence data as inputs to infer the corresponding integration results, e.g., transforming an acceleration sequence to speed variance. Fig. 6 shows the model architecture, taking acceleration data as an example. We sequentially connect four TCN stacks, a slice layer, and an FC layer to produce the output.

1) TCN Stack: Each TCN stack consists of three TCN units and a residual connection (the 1×1 convolution).

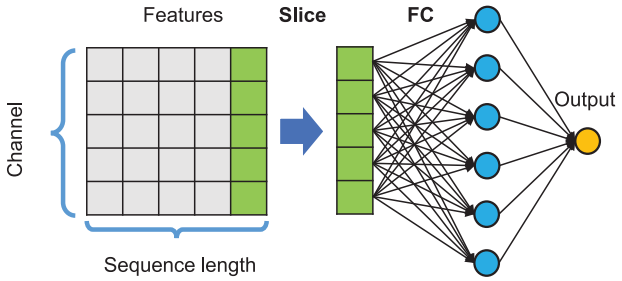


Fig. 7. Slice and FC layers.

TCN Units: In each TCN unit, we sequentially connect a weight normalization (WN), a rectified linear unit (ReLU), and a spatial dropout after each causal dilated convolution. The WN component is used to normalize the weight matrix rather than raw data, thus eliminates the dependence on batch distribution when training sequential data. The spatial dropout component removes local features at a certain probability, thus is used for regularization and avoid overfitting during training.

Receptive Field: The TCN model provides a general inertial sequence learning framework, and its receptive field depends on three factors: 1) the network depth n; 2) filter size k; and 3) dilation factor d. For example, assume the sampling rate of accelerations is 50 Hz and the time window for inference is 5 s, thus an acceleration sequence consists of 250 elements. As shown in Fig. 6, our model consists of four TCN stacks, with dilation factor $d_i = 2^{i-1}$ ($i = \{1, 2, 3, 4\}$ represents the stack index). Each TCN stack contains three dilated causal convolutions (i.e., $n = 3$) with the same dilation factor. We set the filter size $k = 7$, thus our model can involve the first element in the sequence, i.e.,

$$1 + (k-1) \cdot \sum_{i=1}^{4} d_i \cdot n = 1 + (7-1) \cdot \sum_{i=1}^{4} 2^{i-1} \cdot 3 = 271 > 250.$$

Residual Connections: To train such a deep neural network with 15 layers (12 layers of convolutions and three layers of FC), we employ a generic residual connection in each TCN stack. This improves the learning of identity mapping rather than the entire transformation, which has shown to be effective in training deep networks. Different from the standard ResNet [14] which involves the input directly to form the residual block, causal dilated convolutions have changed the size between inputs and outputs, thus we cannot compute the summation as-is. We exploit an additional 1×1 convolution to transform the size of inputs for the elementwise addition operator.

2) Slice and Fully Connected Layers: As shown in Fig. 7, the dimension of TCN features after causal dilated convolutions is determined by the length of a sequence and the number of feature channels (i.e., the size of hidden vectors). We slice the features and only remain ones of the last timestamp. Then, we feed this feature vector to three fully connected (FC) layers to produce the final result.

3) Orientation Inference: We use the angular rate sequence to infer the vehicle's rotation and combine with the predicted velocity to produce the final trajectory. Since the sampling rate of inertial sensors is always fixed, our orientation inference model shares the same design with the velocity. However, since the orientation angle is periodic with a period of 2π, we use the deviation angle $\Delta d_t = d_t - d_0$ as inputs for training.

C. Loss Computation

For training, we use the *SmoothL*1*Loss* as loss metric, i.e.,

$$\text{loss} = \begin{cases} \frac{1}{n}\sum_{i=1}^{n} 0.5 \cdot \left(x_i - \hat{x}_i\right)^2, & \left|x_i - \hat{x}_i\right| \le 1 \\ \frac{1}{n}\sum_{i=1}^{n} \left|x_i - \hat{x}_i\right| - 0.5, & \left|x_i - \hat{x}_i\right| > 1 \end{cases} \tag{12}$$

where $\hat{x}$ and x are the predicted value and corresponding ground truth. n denotes the number of all predictions. This loss metric combines the mean square error (MSE) and the mean absolute error (MAE), thus smooths the loss value from outliers.

VI. Customized Model Training

With the large-scale database collected via crowdsensing, we have established a general location inference model to be used by all users. However, based on our observation, vehicle's tracking accuracy differs with the same pretrained model even on phones with the same type [evaluated in Fig. 17(b)]. This is due to the inconsistent sensor quality, type of vehicles, driving habits, or surrounding environments (e.g., temperature). Thus, training a customized location inference model to fit individual smartphones is critical to improve the tracking accuracy.

A naive and most straightforward approach is to leverage transfer learning, i.e., fine-tuning a pretrained model with customized small amount of data. However, this incurs several problems in our application: 1) it is not efficient to train a deep learning model on mobile-edge devices due to the hardware resource limitations, thus the training is preferred to be moved to cloud backend or offline; 2) individual users may not want to share and upload personal driving data to the backend due to privacy concerns; and 3) as individual's driving habits, vehicles, and road conditions may change over time, the existing model may get stale shortly, thus we need to update the model actively. Although transfer learning on the local device is possible, it brings more computation burden on the devices and we are losing valuable data to improve the common backbone model.

To deal with the above challenges, we adopt a federated learning [15] framework and customize it to our application scenario, which enables mobile phones to collaboratively learn a shared prediction model while keeping all the training data on the device. The basic idea is that the individual's device downloads the current model from backend, improves it by learning from data collected on the phone, and then summarizes the changes as a small focused update (Fig. 8). Then, only the updates to the model instead of data is sent to the backend, where it is averaged with other users' updates and further used to improve the shared global model. To further secure individual's privacy, we add white noise into the update data before uploading to the cloud. By averaging with such updates from all the users, such white noises are canceled to a minimum level, while the major update information is

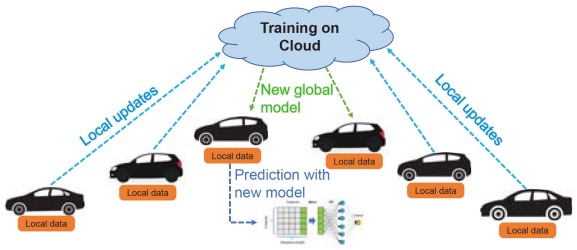


Fig. 8. Architecture of federated learning.

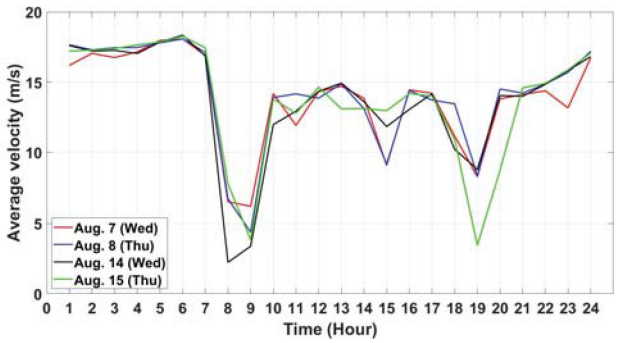


Fig. 9. Traffic velocity variation during one day.

used for updating the shared global model. This gives us the following benefits which solve our problem: it only requires on-device training on very small amount of local data and does not incur privacy issues. Besides, the local training is based on shared global model, which makes it more customized to the individual smartphone while it is getting better generalization capabilities. Note that federated learning is just a general framework, we have to implement all the detailed designs to fit our problem.

Specifically, our training process contains the following steps.

Step 1: Every participated vehicle computes training gradients locally with mini batches when GPS data is available, then the computed gradients instead of raw data are sent to the server cloud.

Step 2: The server performs secure aggregation by averaging all the weights uploaded from vehicles.

Step 3: The server updates the shared global model with aggregated gradients and sends back the updated model to participated vehicles.

Step 4: Each participated vehicle receives the latest shared model and uses it for prediction and customizes the local training process.

As the iterations through the above steps evolve, the shared global model gets better generalization capability without incurring privacy issues. Note that on-device training is only scheduled when devices are in charging mode, minimizing the impact of battery life. Compared to fine-tuning on mobile devices using transfer learning, the federated learning framework enables the improvement of a shared global model, which is beneficial for all users. Meanwhile, the active on-device learning with the latest data enables the customized model for each individual, provides higher accuracy as the driving dynamic changes (evaluated in Section VIII-F).

VII. Auxiliary Attributes

In this section, we involve more auxiliary attributes which can be used to further improve the tracking accuracy.

A. Historical Traffic Speed

Although our TCN model produces vehicle's locations in real time, its velocity and position errors are seriously accumulated during long-time tracking. Thus, we explore an RNN model to yield historical traffic speed via crowdsensing, so as to calibrate the predicted velocity from unbounded errors. Especially, our RNN model is trained beforehand via cloud servers, and used as the preliminary process before TCN learning, thus it will not impede real-time tracking performance.

Our intuition comes from the daily and weekly periodicity on the traffic speed. Fig. 9 depicts traffic velocity variations on a highway road in Beijing during two weeks. The peak values on weekdays are much higher than ones on weekends, and very similar to one day and one week earlier.

Based on the above observation, historical speed information provides an effective hint to calibrate accumulated errors. However, historical trajectories are always missing due to the loss of GPS, especially in tunnels. Thus, we recover such data to produce complete historical speed features at fine-grained areas. Note that at this stage, our inputs include historical inertial sequences as well as anteroposterior GPS trajectories before entering and after leaving GPS blocked areas.

To simplify the problem, we divide it into two subtasks: 1) velocity recovery and 2) orientation recovery. Below, we take the velocity recovery as an example. We aim to recover the missing velocity records by corresponding acceleration sequences and opportunistic velocity observations, e.g., the GPS speed before entering and after leaving the tunnel. The generator f_v can be formalized as

$$\hat{v}_{0:t} = f_v(a_{0:t}, v_{0:t}, m_{0:t}) \tag{13}$$

where $\hat{v}$ denotes the predicted velocity sequence, a denotes the complete acceleration sequence, v is the opportunistic velocity observations from GPS, and m is the corresponding time mask. For example, $m_i = 1$ when the GPS is available, and $v_i = m_i = 0$ otherwise.

Particularly, we adopt a bidirectional long short-term memory (Bi-LSTM) network model (Fig. 10) to overcome the vanishing gradient and exploding gradient problems in RNN. Our bidirectional design captures both forward information before entering and backward information after leaving GPS blocked areas, thus eliminates error accumulations. Note that we add a *short cut connection* directly from the input velocity observation (v_i and m_i), thus our model mainly focuses on fitting the missing data rather than the observed ones. Finally, we use an FC network to predict the missing velocity

$$\hat{v}_t = \mathrm{FC}\left(v_t \circ m_t \circ \overrightarrow{h}_t \circ \overleftarrow{h}_t\right) \tag{14}$$

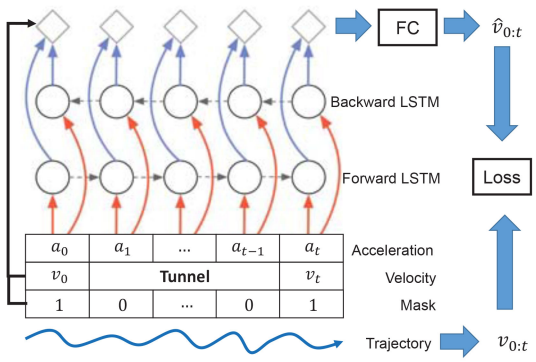


Fig. 10. Bi-LSTM architecture for speed recovery.

where $\overrightarrow{h}_t$ and $\overleftarrow{h}_t$ denote the outputs from forward and backward LSTM cells, which are formalized as

$$\overrightarrow{h}_t = \text{LSTMCell}_f\left(a_t, v_t, m_t, \overrightarrow{h}_{t-1}\right)$$
$$\overleftarrow{h}_t = \text{LSTMCell}_b\left(a_t, v_t, m_t, \overleftarrow{h}_{t+1}\right). \quad (15)$$

During training, we compute MSE as the loss metric

$$\text{loss} = \frac{\sum_t \left(\hat{v}_t^2 - v_t^2\right) \cdot (1 - m_t)}{\sum_t (1 - m_t)}. \quad (16)$$

B. Recent Speed Trend

Besides historical traffic features, we also predict the next speed sequence based on recent speed observations, which are gathered via the last available GPS before losing signal receptions. Different from speed inference by inertial data, we cast speed trend learning as a sequence generative problem, i.e., the joint probability of a speed sequence $v_{1:T}$ is factored as a product of conditional probabilities

$$p(v_{1:T}) = \prod_{t=1}^{T} p(v_t | v_1, \ldots, v_{t-1}). \quad (17)$$

Thus, each element v_t is produced by only previous samples. To ensure efficient deployment on smartphones, we explore a TCN structure for prediction similar to inertial sequence learning. The only difference is the type and length of the input sequence, and we continuously concatenate the last 5 s speed sequences to predict the next value.

C. Social Attributes

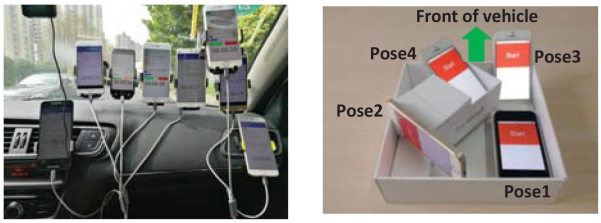
We further implement a Web crawler to extract more social attributes as additional features, including the road map, weather condition, holiday index, and the day in week information. Finally, we directly concatenate the inertial inference with the historical traffic speed one day/week earlier, the recent speed trend, and the embedded social attributes into an FC layer.

VIII. Evaluation

In this section, we evaluate each component of our system and compare with the state-of-the-art methods. We also develop a prototype on the smartphone and evaluate its practical performance in new tunnels.

TABLE I
Data Set Information

Dataset	**Crowdsourced**	**Dedicated**
Source	DiDi platform	Our prototype
Location	Beijing and Shanghai	Beijing and Shanghai
Time	(529 + 716) hours	(23 + 13) hours
Distance	(14326 + 18478) km	(414 + 245) km
Smartphones	(557 + 767) phones	3 phones
Ground truth	GPS (1Hz)	Dedicated IMU (200Hz)



(a) (b)

Fig. 11. Data collection by crowdsensing and a mold for phone pose estimation. (a) Crowdsensing. (b) Mold with four phones.

A. Methodology

Crowdsourced/Dedicated Data Set: Our large-scale real-world crowdsourced data set is collected by the DiDi platform in Beijing and Shanghai, the largest two cities in China with millions of vehicles. The time period is from December 8, 2019 to January 9, 2020. The traffic data is crowdsourced from 557 phones in Beijing covering 14 326 km distances during 529 h, and from 767 phones in Shanghai covering 18 478 km distances during 716 h, respectively.

In addition, in order to test our real-time performance in new tunnels, we develop and install the prototype on three Android phones (Xiaomi Mi Note, Huawei P9, and Oppo R9m) with the Tensorflow Mobile interface for model transplantation. We further collect data over six days in Beijing and Shanghai by ourselves. The ground truth for evaluation is measured by a dedicated highly accurate IMU device (XW-GI7660 from StarNeto company).

Details for our data sets are shown in Table I.

Training Details: When vehicles are driven outdoors, their GPS locations are recorded as the ground truth to train our tracking model, which transforms inertial sequences to trajectories. Note that although IMU and GPS are different sensors, they are both embedded inside commodity smartphones thus their relative displacement (i.e., the lever arm) is very small.

In the implementation, our TCN and RNN models are both implemented in PyTorch, and trained for 100 epochs using the Adam optimizer [16] with learning rate of $3e-4$. The ratio for training/validation/test split is 6:2:2. We set the TCN batch size of 32 and dropout of 0.2, and the RNN batch size 256 and dropout of 0.5. The RNN model is trained beforehand on cloud servers and produces historical traffic speeds one day/week earlier to calibrate the real-time tracking from TCN.

Evaluation Metrics: We collect over 80 000 tunnel trajectories within one month by DiDi platform among China, and Fig. 12 depicts their length distribution. In addition, since the average speed in tunnels is around 16 m/s (nearly 60 km/h),

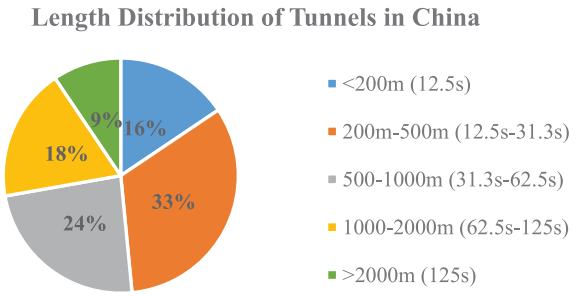


Fig. 12. Tunnel length distribution among China, collected by DiDi ride-hailing platform.

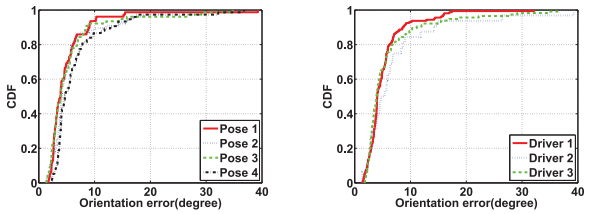


Fig. 13. Pose estimation errors. (a) Four different poses. (b) Three different vehicles and drivers.

the average and 70-percentile time durations to pass a tunnel are 30 and 60 s, respectively. Thus, such durations are exploited to evaluate the tracking accuracy.

Besides calculating location errors, we also evaluate vehicle's velocity and orientation errors along the trajectory. Since we exploit outdoor GPS locations as the ground truth to train our model, the vehicle's velocity and orientation are both defined on 2-D maps, i.e., its horizontal speed on the ground and its relative rotation along the gravity direction.

Comparison Benchmarks: We compare our performance with EKF, LSTM, and GRU solutions for vehicle dead reckoning. The EKF approach is currently applied by many ride-hailing platforms to track vehicles in tunnels. The LSTM is implemented based on IONet [4] which is designed for pedestrian tracking, and GRU [10] is known to be much faster than LSTM.

B. Evaluation of Individual Components

1) Pose Estimation: We design a mold to fix four phones with different poses: horizontal, lean, vertical, and box [Fig. 11(b)]. The mold is placed flat inside the vehicle. We measure each pose's attitude in the mold as the ground truth.

Fig. 13(a) shows the pose estimation errors with different placements in the mold. We observe that the 90-percentile error is around 9°, and the largest error is less than 40°. Fig. 13(b) presents the effect of vehicles and drivers. Despite different drivers and cars, we achieve similar pose estimation accuracy, also around 9° at 90-percentile.

Next, we estimate the probability of the forward and backward directions of the vehicle, $p(\theta = \theta_2)$ and $p(\theta = \theta_2)$. We calculate the distribution of the estimated probability of the backward direction, i.e., the estimation error. As shown in Fig. 14, it achieves more than 99% accuracy at 90 percentile and a maximum error of 23%.

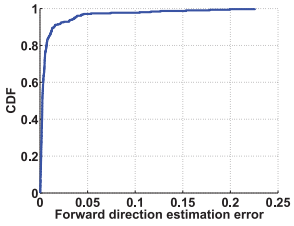


Fig. 14. Distribution of estimated probability of the backward direction.

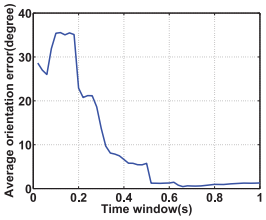


Fig. 15. Pose estimation errors with different time windows.

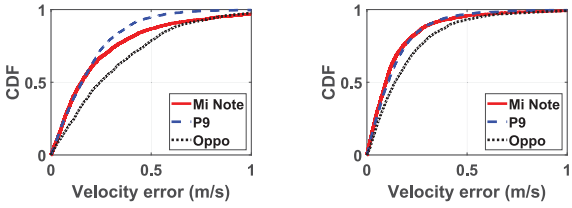


Fig. 16. Velocity inference errors for three phones in (a) Beijing and (b) Shanghai database, respectively.

Fig. 15 shows the pose estimation accuracy with different time windows. We observe that the 90-percentile pose estimation error is reduced when increasing the time window. Additionally, the orientation errors stay stable when the time window is larger than 0.5 s, which reflects the real-time performance of our system.

2) Acceleration Sequence Learning: We evaluate the velocity error of our acceleration sequence learning component. We set the time window as 5 s, i.e., producing current velocity based on the last 5 s accelerations. The learning rate is set as $3e-4$. The CDF curves are shown in Fig. 16(a) on the Beijing database and Fig. 16(b) on the Shanghai database, respectively. We have evaluated three types of smartphones in both databases, i.e., Xiaomi Mi Note, Huawei P9, and Oppo R9m. We find that: 1) our acceleration sequence modeling is very accurate, with the 80th-percentile error at 0.3–0.5 m/s in Beijing and 0.2–0.3 m/s in Shanghai, respectively. This yields accurate vehicle tracking in real time and 2) the Mi Note and P9 phones have similar accuracy whereas the Oppo phone has the largest errors in both cases, this indicates that the Oppo phone is possibly equipped with a less precise accelerometer.

3) Rotation Sequence Learning: Fig. 17(a) depicts the orientation errors for rotation sequence learning with three types of smartphones in Beijing (similarly in Shanghai). The time window is set as 60 s, i.e., inferring current orientation based on the last 60 s gyroscope readings. We observe that vehicle orientation errors are relatively small, with the 80th-percentile

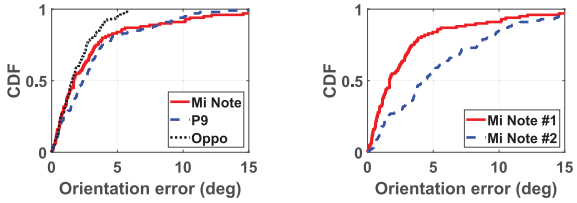


Fig. 17. Orientation errors for three types of phones in Beijing, and comparison between the same type. (a) Beijing database. (b) Two Mi Note phones.

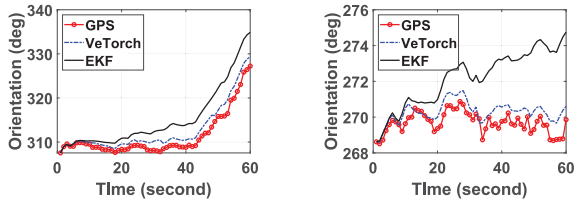


Fig. 18. Orientation errors during 60 s. GPS speed direction is the ground truth. (a) Sample trajectory in Beijing. (b) Sample trajectory in Shanghai.

TABLE II
EVALUATION ON MISSING DATA RECOVERY

	Velocity	Orientation
MAE	0.219m/s	0.819 degrees
RMSE	0.324m/s	1.833 degrees

error around 3° to 5°. Both the average and maximum errors of the Oppo phone are the least, which indicates it may be equipped with a high-quality gyroscope.

We further compare different phones of the same type. Fig. 17(b) shows the orientation errors for two Mi Note phones. We find that orientation errors vary significantly even for the same type of phone, i.e., the 80th-percentile error differs from 5° to 10°. This is possibly due to the inconsistent quality of its gyroscope in different environments.

Fig. 18 depicts vehicle orientation errors on two sample trajectories during 60 s, in Beijing and Shanghai, respectively. Compared with the ground truth, the accumulated errors of EKF are extremely large due to noises of the gyroscope, and our method can reduce long-term orientation errors to a much smaller value (close to the ground truth).

4) History Recovery: Given anteroposterior GPS speeds before entering and after leaving the GPS blocked environments, our Bi-LSTM model learns both the forward and backward information, thus significantly reduces error accumulations when inferring missing trajectories. Fig. 19(a) and (b) shows two examples on vehicle velocity and orientation recovery, respectively. We can observe that the maximum errors appear at the middle of this sample trajectory, while both the head and the tail of recovered results match the ground truth precisely. Table II further depicts the MAE and RMSE for both velocity recovery and orientation recovery, which shows much lower errors than speed inference.

5) Speed Trend Modeling: To model recent speed trend, we set the time window as 5 s, and measure vehicle velocity errors

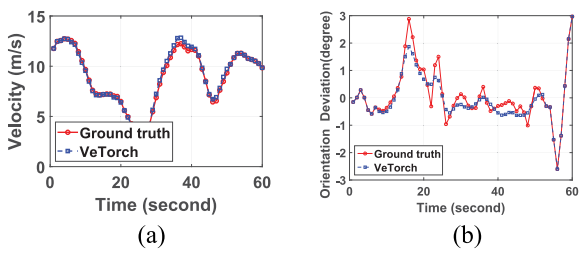


Fig. 19. Examples on (a) velocity and (b) orientation recovery. The time window is 1 min.

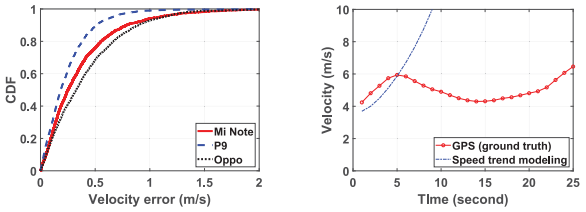


Fig. 20. Velocity errors for vehicle trend modeling. The time window is 5 s. (a) Beijing database. (b) Sample trajectory.

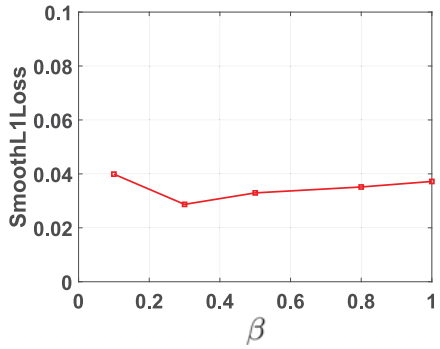


Fig. 21. SmoothL1Loss for hyperparameter β.

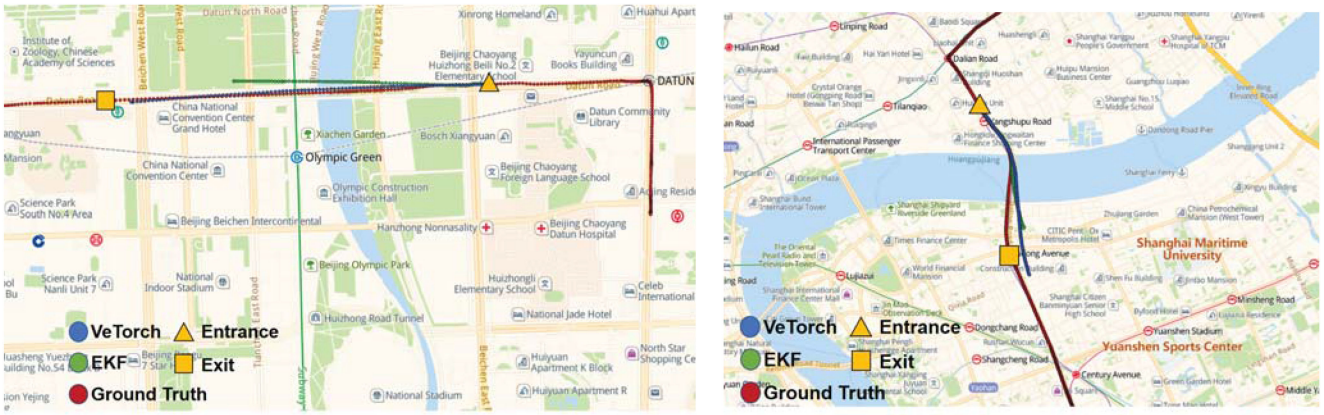
via only speed trend modeling. As shown in Fig. 20(a), the 80th-percentile velocity errors on three types of smartphones are 0.5, 0.4, and 0.7 m/s, respectively. Compared with acceleration sequence modeling, the above errors are at least 2× more. The reason is simple: speed trend only uses previous data and cannot capture current driving actions.

Fig. 20(b) further illustrates long-term tracking effects of speed trend modeling. Compared with the ground truth, velocity predictions from speed trend are relatively accurate during the early period in the tunnel. Then, the vehicle slows down but the speed trend model still increases the predictions.

6) Hyperparameter: To show the effectiveness of the combination coefficient β, we change it from 0.1 to 1 and calculate the SmoothL1Loss of our model. The result is shown in Fig. 21. We find that our model is robust for a wide range of β, and the least loss is acquired when $\beta = 0.3$.

C. Comparison With Others

Table III depicts vehicle's location errors during 30 and 60 s intervals, in Beijing and Shanghai, respectively. We use MAE as the localization metric. Compared with EKF, LSTM



(a) (b)

Fig. 22. Examples of real-time tracking in new tunnels, with our prototype and the existing EKF solution, respectively. (a) Tunnel in Beijing. (b) Tunnel in Shanghai.

TABLE III
MAE ON LOCATION INFERENCE (M)

Method	30-second tracking		60-second tracking	
	Beijing	Shanghai	Beijing	Shanghai
EKF	87.88	59.39	144.24	99.98
LSTM (IONet)	38.25	33.29	82.51	87.70
GRU	35.70	31.73	76.18	69.85
VeTorch	**33.92**	**29.07**	**67.57**	**68.24**

(IONet [4]), and GRU [10], we achieve the least MAE values. The reason for large errors on LSTM and GRU is that they drop previous inertial items over a long-term sequence, but each inertial item is important for location inference.

D. Real Time Performance in New Tunnels

In order to evaluate the real-time performance in new tunnels, we have developed a prototype on three phones with the Tensorflow Mobile interface, and tested our prototype in new tunnels in Beijing and Shanghai, respectively. The tunnels remain consistently unknown to both the EKF solution and our method, thus vehicles can only employ current inertial data to infer real-time locations when driving in tunnels (between the entrance and the exit). Ground truth is measured via a dedicated and highly accurate IMU device (XW-GI7660 from StarNeto company).

Fig. 22 demonstrates our effectiveness for real-time tracking in new tunnels. Compared with the EKF, our method performs more accurate trajectories on both distance and orientation. Fig. 23 further depicts vehicle's real-time location errors along the example tunnel trajectory in Beijing, with 18.32 m errors after 95 s driving.

In addition, we also measure smartphone performances on Huawei P9, with four tracking models (elaborated in Table IV). The input data contains 1920 inertial sequences, and each sequence lasts for 1 min. The storage of our model is 2.25 MB, slightly larger than the other three but still remains negligible for smartphones. The average inference time on one sequence is 7 ms for our model, almost half compared with LSTM and GRU. The average power for our model is also lower than

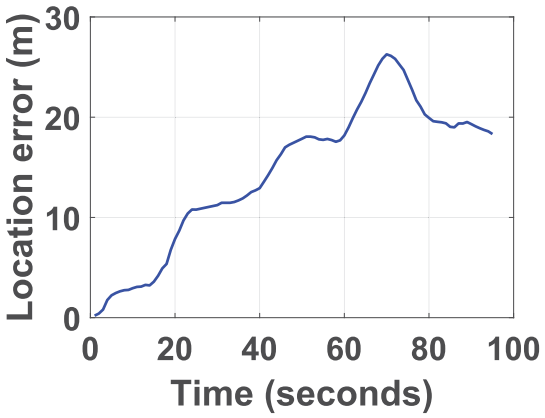


Fig. 23. Vehicle tracking with our prototype in a tunnel in Beijing.

TABLE IV
PROTOTYPE PERFORMANCE ON SMARTPHONES

Method	Storage	Inference Time	Average Power
EKF	0.05MB	1ms	24415uW
LSTM	1.92MB	17ms	139792uW
GRU	1.65MB	13ms	99714uW
VeTorch	2.25MB	7ms	78682uW

them. Thus, our model is much more efficient than LSTM and GRU on smartphones.

E. Long Time Tracking

Although majority tunnel trajectories are within less than 2 min (nearly 90-percentile in Fig. 12), we also track a 5-min trajectory in Beijing to illustrate how our method performs in extreme long tunnels. As Fig. 24 shows, the maximum location error is only 31.7 m for this long trace. Fig. 25 further depicts its velocity errors during 5 min, and we observe that the predicted velocity by VeTorch is slightly lower than the ground truth when driving at a fast speed.

F. Customized Model Training

We evaluate our federated learning-based local customized model training mechanism by updating the shared model with

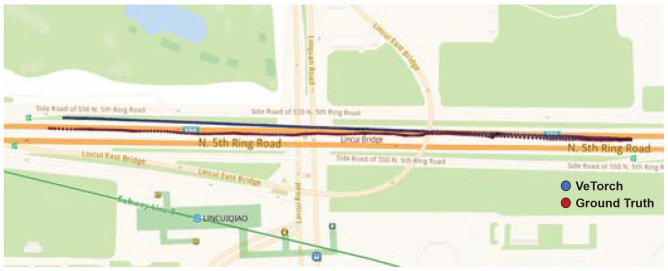


Fig. 24. Tracking vehicles in an extreme long tunnel for 5 min.

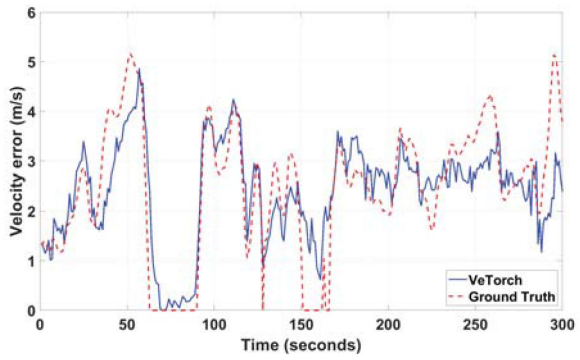


Fig. 25. Predicted velocity errors by VeTorch during 5 min.

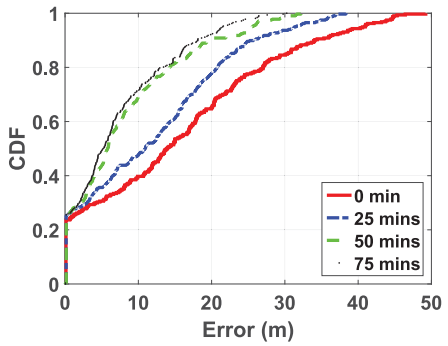


Fig. 26. Vehicle's prediction location errors with different amounts of latest training data.

different amounts of latest data, when both inertial and GPS signals are available. A pretrained model is used as the baseline and we use 25, 50, and 75 min data captured during runtime to actively train the baseline model locally, and evaluate the performance with the next 25 min' data. We evaluate the location error every 5 s and draw the accumulated errors in Fig. 26. Without active learning with latest training data, the 80-percentile error is larger than 25 m and the maximum error is ∼50 m. With 25 min' latest data training, the 80-percentile error decreases to 20 m. The error is further reduced as we use larger time window data (50 and 75 min) for active training. We choose 50 min as the time window in our design as it achieves the best balance of accuracy and local training resource consumption. Note that about 20% of the error are close to zero, this is because vehicles are in nonmoving mode (waiting for traffic lights or in parking mode).

IX. Related Work

A. Intelligent Transportation Systems

Intelligent transportation systems (ITS) improve the efficiency of daily transport and provide innovative services to traffic management. Among other applications, traffic speed prediction (TSP) always plays a fundamental role. There are generally two kinds of method for TSP: 1) the parametric solutions, e.g., ARIMA [17] which models the traffic in a stationary process and 2) the nonparametric solutions, e.g., SVR [18] and LR [19] which formulate it as a regression problem. Currently, convolutional neural networks (CNNs)-based models are used for TSP via large-scale historical traffic data [20], [21].

Travel time estimation (TTE) is also important to location-based services for vehicles. Existing TTE efforts can be classified into two categories: 1) the route-based approaches and 2) the data-driven approaches. The first [20], [22], [23] calculates the total travel time as the time summation on each road segment and intersection. The second [24]–[28] formulates it as a multivariate time-series prediction problem.

B. Sequential Deep Learning

RNNs are dedicated sequential learning models and have been widely used in natural language processing [29], machine translation [30], and speech recognition [31]. LSTM [9] and GRU [10] can capture temporal dynamics on both local information and long-term dependency along sequence, thus become the most popular RNN structures. Attention mechanism [32] is also widely applied to capture the weights of a sequence.

CNNs have also been used for sequence processing, especially on sentence classification [33], [34], language modeling [35], and audio synthesis [12]. Recent work indicates that CNN performs better than RNN when processing sequences with long-term memory. TCN [11] uses 1-D causal dilated convolutions over the sequence and obtains wide receptive fields. Some latest work [36], [37] establish a structural bridge that incorporate techniques from both RNN and CNN.

C. Inertial Tracking and Monitoring

Inertial dead-reckoning techniques have been widely used for indoor tracking. Pedestrian step counting can be easily detected via accelerations [38], but stride lengths and heading directions are usually difficult to estimate. Recent research fuse with other reference signals for periodically calibration thus mitigate the error accumulation, e.g., UnLoc uses virtual indoor landmarks, MapCraft [39] uses the floor plan, Hilsenbeck *et al.* used WiFi fingerprinting [40], and some use ambient magnetic fields [41], [42]. However, they all rely on the step counting results which are improper for vehicles. IONet [4] is the first to abandon step counting and propose a direct inertial odometry, and MotionTransformer [5] enhances it by domain-invariant features, but they both aim to track the pedestrian. VeTrack [43] is designed for vehicle tracking with low driving speed and plenty of landmarks (e.g., bumps and turns) which are not common in tunnels.

Besides inertial tracking, there are many research using smartphones' inertial data to monitor dangerous driving behaviors (e.g., dangerous driving alert [44] and CarSafe [45]); identify traffic accidents (e.g., Nericell [46] and WreckWatch [47]); sense driver phone use (e.g., car speaker [48]); and inspect the road anomaly or conditions (e.g., Pothole Patrol [49]). The vehicle's GPS speed is a critical factor to such applications, but the signal can be weak or even unavailable in many GPS blocked environments such as tunnels.

X. Conclusion

In this article, we propose *VeTorch* which tracks a vehicle's location in real time, with only smartphone's inertial data. It opens up the possibility to ubiquitous location-based services for vehicles in GPS blocked environments. *VeTorch* devises a novel inertial sequence learning framework for fast, secure, and active deployment on phones. Extensive experiments in large-scale real-world data sets from a modern ride-hailing platform and our prototype have demonstrated the effectiveness compared with the state of the art.

References

[1] M. Kotaru, K. Joshi, D. Bharadia, and S. Katti, "SpotFi: Decimeter level localization using WiFi," in *Proc. ACM Conf. Spec. Interest Group Data Commun. (SIGCOMM)*, 2015, pp. 269–282.

[2] D. Vasisht, S. Kumar, and D. Katabi, "Decimeter-level localization with a single WiFi access point," in *Proc. 13th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2016, pp. 165–178.

[3] Y. Zhuang, J. Yang, Y. Li, L. Qi, and N. El-Sheimy, "Smartphone-based indoor localization with bluetooth low energy beacons," *Sensors*, vol. 16, no. 5, p. 596, 2016.

[4] C. Chen, X. Lu, A. Markham, and N. Trigoni, "IONet: Learning to cure the curse of drift in inertial odometry," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 6468–6476.

[5] C. Chen *et al.*, "MotionTransformer: Transferring neural inertial tracking between domains," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 8009–8016.

[6] L. Chang, F. Zha, and F. Qin, "Indirect Kalman filtering based attitude estimation for low-cost attitude and heading reference systems," *IEEE/ASME Trans. Mechatronics*, vol. 22, no. 4, pp. 1850–1858, Aug. 2017.

[7] I. T. Jolliffe, *Principal Component Analysis*. New York, NY, USA: Springer, 2002.

[8] Y. Wang, J. Yang, H. Liu, Y. Chen, M. Gruteser, and R. P. Martin, "Sensing vehicle dynamics for determining driver phone use," in *Proc. ACM 11th Annu. Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2013, pp. 41–54.

[9] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[10] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014. [Online]. Available: arXiv:1409.1259.

[11] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," 2018. [Online]. Available: arXiv:1803.01271.

[12] A. van den Oord *et al.*, "WaveNet: A generative model for raw audio," 2016. [Online]. Available: arXiv:1609.03499.

[13] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin, "A convolutional encoder model for neural machine translation," 2016. [Online]. Available: arXiv:1611.02344.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, 2016, pp. 770–778.

[15] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," 2016. [Online]. Available: arXiv:1610.05492.

[16] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: arXiv:1412.6980.

[17] B. M. Williams and L. A. Hoel, "Modeling and forecasting vehicular traffic flow as a seasonal ARIMA process: Theoretical basis and empirical results," *J. Transp. Eng.*, vol. 129, pp. 664–672, 2003.

[18] C.-H. Wu, J.-M. Ho, and D. T. Lee, "Travel-time prediction with support vector regression," *IEEE Trans. Intell. Transp. Syst.*, vol. 5, no. 4, pp. 276–281, Dec. 2004.

[19] G. Ristanoski, W. Liu, and J. Bailey, "Time series forecasting using distribution enhanced linear regression," in *Advances in Knowledge Discovery and Data Mining*. Heidelberg, Germany: Springer, 2013, pp. 484–495.

[20] J. Wang, Q. Gu, J. Wu, G. Liu, and Z. Xiong, "Traffic speed prediction and congestion source exploration: A deep learning method," in *Proc. 16th IEEE Int. Conf. Data Min. (ICDM)*, 2016, pp. 499–508.

[21] X. Ma, Z. Tao, Y. Wang, H. Yu, and Y. Wang, "Long short-term memory neural network for traffic speed prediction using remote microwave sensor data," *Transp. Res. C, Emerg. Technol.*, vol. 54, pp. 187–197, May 2015.

[22] R. Sevlian and R. Rajagopal, "Travel time estimation using floating car data," 2010. [Online]. Available: https://arxiv.org/abs/1012.4249.

[23] B. Pan, U. Demiryurek, and C. Shahabi, "Utilizing real-world transportation data for accurate traffic prediction," in *Proc. IEEE 12th Int. Conf. Data Min.*, Brussels, Belgium, 2012, pp. 595–604.

[24] H. Wang, X. Tang, Y.-H. Kuo, D. Kifer, and Z. Li, "A simple baseline for travel time estimation using large-scale trip data," in *Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2016, pp. 1–4.

[25] D. Wang, J. Zhang, W. Cao, J. Li, and Y. Zheng, "When will you arrive estimating travel time based on deep neural networks," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 2500–2507.

[26] H. Zhang, H. Wu, W. Sun, and B. Zheng, "DeepTravel: A neural network based travel time estimation model with auxiliary supervision," in *Proc. 27th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2018, pp. 3655–3661.

[27] Y. Li, K. Fu, Z. Wang, C. Shahabi, J. Ye, and Y. Liu, "Multi-task representation learning for travel time estimation," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery Data Min. (KDD)*, 2018, pp. 1695–1704.

[28] Z. Wang, K. Fu, and J. Ye, "Learning to estimate the travel time," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Min. (KDD)*, 2018, pp. 858–866.

[29] R. Collobert and J. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," in *Proc. ACM 25th Int. Conf. Mech. Learn. (ICML)*, vol. 307, 2008, pp. 160–167.

[30] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning," 2015. [Online]. Available: https://arxiv.org/abs/1511.01432.

[31] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proc. ACM 31st Int. Conf. Mech. Learn. (ICML)*, vol. 32, 2014, pp. 1764–1772.

[32] Y. Liang, S. Ke, J. Zhang, X. Yi, and Y. Zheng, "GeoMAN: Multi-level attention networks for geo-sensory time series prediction," in *Proc. 27th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2018, pp. 3428–3434.

[33] N. Kalchbrenner, E. Grefenstette, and P. Blunsom, "A convolutional neural network for modelling sentences," in *Proc. 52nd Annu. Meeting Assoc. Comput. Linguist. (ACL)*, 2014, pp. 655–665.

[34] Y. Kim, "Convolutional neural networks for sentence classification," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1746–1751.

[35] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Annu. Conf. Neural Inf. Process. Syst. (NIPS)*, 2014, pp. 3104–3112.

[36] J. Donahue *et al.*, "Long-term recurrent convolutional networks for visual recognition and description," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2015, pp. 2625–2634.

[37] S. Bai, J. Z. Kolter, and V. Koltun, "Trellis networks for sequence modeling," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–18.

[38] Y. Shu, K. G. Shin, T. He, and J. Chen, "Last-mile navigation using smartphones," in *Proc. ACM 21st Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2015, pp. 512–524.

[39] Z. Xiao, H. Wen, A. Markham, and N. Trigoni, "Lightweight map matching for indoor localisation using conditional random fields," in *Proc. 13th Int. Symp. Inf. Process. Sens. Netw. (IPSN)*, 2014, pp. 131–142.

[40] S. Hilsenbeck, D. Bobkov, G. Schroth, R. Huitl, and E. G. Steinbach, "Graph-based data fusion of pedometer and WiFi measurements for mobile indoor positioning," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput. (UbiComp)*, 2014, pp. 147–158.

[41] S. Wang, H. Wen, R. Clark, and N. Trigoni, "Keyframe based large-scale indoor localisation using geomagnetic field and motion pattern," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Daejeon, South Korea, 2016, pp. 1910–1917.

[42] S. Shen, M. Gowda, and R. R. Choudhury, "Closing the gaps in inertial motion tracking," in *Proc. 24th Annu. Int. Conf. Mobile Comput. Netw. (MobiCom)*, 2018, pp. 429–444.
[43] R. Gao, M. Zhao, T. Ye, F. Ye, Y. Wang, and G. Luo, "Smartphone-based real time vehicle tracking in indoor parking structures," *IEEE Trans. Mobile Comput.*, vol. 16, no. 7, pp. 2023–2036, Jul. 2017.
[44] J. Lindqvist and J. Hong, "Undistracted driving: A mobile phone that doesn't distract," in *Proc. ACM 12th Workshop Mobile Comput. Syst. Appl. (HotMobile)*, 2011, pp. 70–75.
[45] C.-W. You *et al.*, "CarSafe app: Alerting drowsy and distracted drivers using dual cameras on smartphones," in *Proc. ACM 11th Annu. Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2013, pp. 461–462.
[46] P. Mohan, V. N. Padmanabhan, and R. Ramjee, "Nericell: Using mobile smartphones for rich monitoring of road and traffic conditions," in *Proc. 6th ACM Conf. Embedded Netw. Sens. Syst. (SenSys)*, 2008, pp. 357–358.
[47] J. White, C. Thompson, H. Turner, B. Dougherty, and D. C. Schmidt, "WreckWatch: Automatic traffic accident detection and notification with smartphones," *Mobile Netw. Appl.*, vol. 16, no. 3, pp. 285–303, 2011.
[48] J. Yang *et al.*, "Sensing driver phone use with acoustic ranging through car speakers," *IEEE Trans. Mobile Comput.*, vol. 11, no. 9, pp. 1426–1440, Sep. 2012.
[49] J. Eriksson, L. Girod, B. Hull, R. Newton, S. Madden, and H. Balakrishnan, "The pothole patrol: Using a mobile sensor network for road surface monitoring," in *Proc. ACM 6th Int. Conf. Mobile Syst. Appl. Services (MobiSys)*, 2008, pp. 29–39.

Ruipeng Gao received the B.S. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2010, and the Ph.D. degree from Peking University, Beijing, in 2016.

He was a Visiting Scholar with Purdue University, West Lafayette, IN, USA, in 2019. He is currently an Associate Professor with the School of Software Engineering, Beijing Jiaotong University, Beijing. He is also with the State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, China. His research interests include mobile computing and applications, Internet of Things, and intelligent transportation systems.

Xuan Xiao received the B.S. degree in network engineering from China University of Mining and Technology, Xuzhou, China, in 2016. He is currently pursuing the Ph.D. degree in software engineering with Beijing Jiaotong University, Beijing, China.

His research interests include mobile computing and applications and Internet of Things.

Shuli Zhu received the B.S. degree in software engineering from Beijing Jiaotong University, Beijing, China, in 2019, where he is currently pursuing the M.S. degree in software engineering.

His research interests include urban positioning and mobile computing.

Weiwei Xing received the B.S. and Ph.D. degrees in computer science from Beijing Jiaotong University, Beijing, China, in 2001 and 2006, respectively.

She is currently a Professor with the School of Software Engineering, Beijing Jiaotong University. Her current research interests include software engineering and intelligent information processing.

Chi Li received the B.S. degree from Nanchang Hangkong University, Nanchang, China, in 2014, and the M.S. degree from Beihang University, Beijing, China, in 2017.

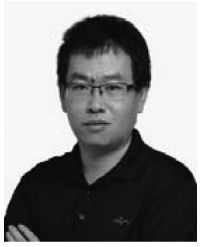
He is currently a Location Algorithm Engineer with DiDi Company, Beijing. His research interests include inertial navigation, integrated navigation, and machine learning.

Lei Liu received the B.S. degree from Shandong University, Jinan, China, in 2010, and the M.S. degree from the University of Chinese Academy of Sciences, Beijing, China, in 2013.

He is currently an Expert Algorithm Engineer with DiDi Company, Beijing, in charge of positioning and POI recommendation.

Li Ma received the M.S. degree from Xi'an Jiaotong University, Xi'an, China, in 2009.

She is currently a Chief Algorithm Engineer with DiDi Company, Beijing, China, in charge of positioning, pickup and drop-off spots suggestion, origin and destination address recommendation and search, with rich experience in data mining, NLP, and machine learning.



Hua Chai received the B.S. and M.S. degrees in computer science from Tianjin University, Tianjin, China, in 2006 and 2009, respectively.

He is currently the General Manager of DiDi Maps and Public Transportation Department, DiDi Company, Beijing, China. He has rich experience in maps technology, Internet advertising, large complex distributed systems (including storage, computing, and scheduling), big data, and machine learning.