Latency Prediction for Delay-sensitive V2X Applications in Mobile Cloud/Edge Computing Systems

Wenhan Zhang[1], Mingjie Feng[1], Marwan Krunz[1], and Haris Volos[2]

[1]Dept. Electrical & Computer Engineering, University of Arizona, Tucson, AZ,
{wenhanzhang, mingjiefeng, krunz}@email.arizona.edu
[2]Silicon Valley Innovation Center, DENSO International America, Inc., San Jose, CA,
haris_volos@denso-diam.com

***Abstract*—Mobile edge computing (MEC) is a key enabler of delay-sensitive vehicle-to-everything (V2X) applications. Determining where to execute a task necessitates accurate estimation of the offloading latency. In this paper, we propose a latency prediction framework that integrates machine learning and statistical approaches. Aided by extensive latency measurements collected during driving, we first preprocess the data and divide it into two components: one that follows a trackable trend over time and the other that behaves like random noise. We then develop a Long Short-Term Memory (LSTM) network to predict the first component. This LSTM network captures the trend in latency over time. We further enhance the prediction accuracy of this technique by employing a k-medoids classification method. For the second component, we propose a statistical approach using a combination of Epanechnikov Kernel and moving average functions. Experimental results show that the proposed prediction approach reduces the prediction error to half of a standard deviation (STD) of the raw data.**

I. Introduction

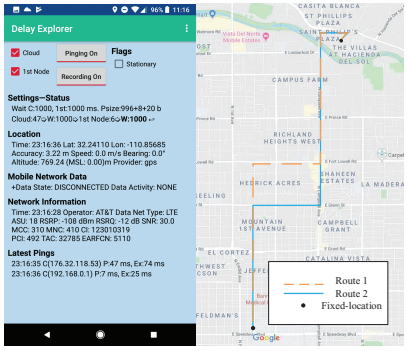
As a key component of future intelligent transportation systems (ITS), connected and autonomous vehicles (CAVs) have attracted intensive research from both academia and industry. Typical forms of CAV communications include vehicle-to-vehicle (V2V), vehicle-to-infrastructure (V2I), vehicle-to-network (V2N), and vehicle-to pedestrian (V2P), which are collectively referred to as vehicle-to-everything (V2X) [1]. V2X enhances the situational awareness of vehicles, facilitating both beyond line of sight (BLOS) safety applications, such as accident/merge alerts and collision prevention, as well as non-safety applications, such as cruise control, multimedia services, and self-parking, among others.

In many instances, V2X applications require executing low-latency yet computationally intensive tasks. For example, objects recognition from images (e.g., pedestrians, other vehicles, etc.) often involves extensive, real-time processing by a deep convolutional neural network [10]. Such processing would be quite prohibitive for the in-vehicle embedded processor [2]. One approach for this challenge is to offload V2X computational tasks to a remote cloud server. However, due to network dynamics and connection intermittency, the end-to-end communication latency between the vehicle and the cloud server can be excessively high. Alternatively, the task may be offloaded to a mobile edge computing (MEC) server [3], [4]. Exploiting MEC servers that are close to base stations (BS) or roadside units (RSUs) results in significant reduction in the communication latency. At the same time, these servers are unlikely to have the same computational resources as a cloud server. To harness the benefits of both MEC and cloud servers, the offloading decision should be made adaptively according to the properties of a task.

Determining where to execute a V2X task necessitates accurate prediction of both its communication and computing latencies. While the computing latency can be estimated based on task size and the computational resources of the MEC/cloud server, estimating the communication latency is more challenging due to the uncertainty of the wireless environment. In most existing works, such latency is approximated by the data transmission time [5], [6], obtained from the task size and data rate. Such approximation overlooks the channel access latency and the round-trip time (RTT) between a vehicle and a MEC/cloud server. In reality, as shown in our measurements, the RTT is quite significant and is also highly dynamic [7]. Unlike data transmission latencies which can be estimated a priori, there is no obvious relationship between the access latency and the system parameters. Due to the dynamics of the wireless channel, the access latency varies rapidly over time. To accurately capture the underlying pattern of the access latency as well as its instantaneous randomness, intelligent exploitation of historical data is required.

The main contributions of this paper are as follows. We first develop a smartphone app called *Delay Explorer* and implement it on a Google Pixel 2 phone. Using this app, we then collect thousands of traces of real-time RTT data in different locations over a period of four months. By applying an appropriate filter, we divide the raw data into

This research was supported in part by NSF (grants CNS-1910348 and CNS-1813401) and by the Broadband Wireless Access & Applications Center (BWAC). Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the author(s) and do not necessarily reflect the views of NSF.



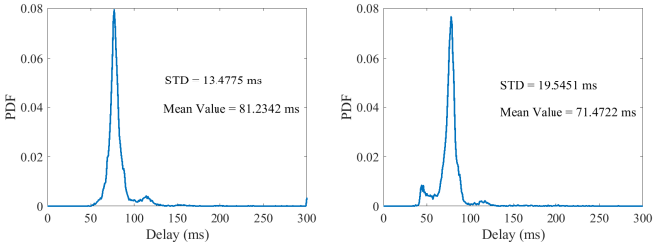
(a) (b)

Fig. 1. (a) Interface of *Delay Explorer* app. (b) locations and routes of our measurements.

a trackable pattern (trend) and its residual. Using the k-medoids algorithm, we cluster the data of the first component into multiple sets according to the signal strength. An extensively trained Long Short-Term Memory (LSTM) network is then designed for each cluster, allowing for predicting the future values of the trend component. For the second component, we propose a hybrid statistical approach, based on a combination of Epanechnikov Kernel function-based probabilistic sampling and moving average function-based prediction. Performance evaluation shows 50% reduction in the prediction error compared to a sample-mean predictor. By decomposing the original problem, we increase the accuracy of each part prominently, e.g., achievable LSTM root mean square error (RMSE) during training decreases from 40% to 10%. Moreover, data points with very high latency can be detected when we include the signal strength parameters into the designed LSTM network. By adding classifiers into the network, clustered data are trained with respective feature groups to further reduce RMSE.

II. Latency and Measurements

To collect real data for the communication latency between a vehicle and MEC/cloud servers, we developed a smartphone app called *Delay Explorer*. This app sends a stream of ping packets to the IP addresses of the edge node and the cloud node, respectively. The ping messages are sent via an AT&T LTE network. The interval between two consecutive messages is set to 500 ms. Ping is a typical measure of the network response latency between a user and a server, and is a key indicator for many delay-sensitive applications, e.g., multi-player online video games. Because MEC servers are typically located close to mobile users (e.g., collocated with BS), the IP address of the first node that responds to the ping message can be regarded as the location of MEC server. In our measurements, the cloud node is the Amazon Web Server (AWS), located at the Amazon cloud service center. *Delay Explorer* also records other system parameters besides RTT, including received signal strength indicators (RSSI), velocity, GPS information, and others. In our prediction framework, we use LTE signal strength (SS), reference signal received power (RSRP), and reference signal received quality (RSRQ)



(a) PDF of cloud node latency (b) PDF of edge node latency

Fig. 2. Distributions of cloud-node latency and edge-node latency based on all measurements.

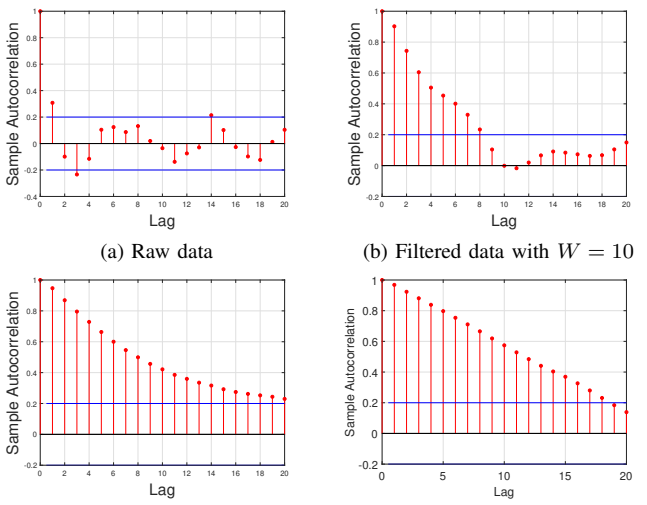
indicators to help classify the signal and predict the latency. These parameters will be explained later. Fig. 1(a) shows the interface of *Delay Explorer*.

Our measurements were taken at fixed locations and also in mobile scenarios. More than 1,600,000 data points were collected. The fixed-location scenarios include an office and an apartment (see Fig. 1(b)). For the mobile scenario, latency measurements were taken while driving along the route in Fig. 1(b). These measurements represent realistic vehicular conditions, whereby node and traffic densities as well as channel conditions change along the route. Furthermore, data collection was conducted over different time periods (including weekdays, weekends, mornings, afternoons, etc.), thus providing a good representation of many practical use cases. The measurements are divided into a training part and a testing part, with the latter part used to evaluate the performance of the proposed predictors. Fig. 2 shows the distributions and statistics of MEC and cloud nodes' RTTs. As expected, the edge node has a lower average RTT than the cloud node, but it has a higher standard deviation (STD) for its RTTs.

III. Overview of Prediction Framework

Our latency measurements show significant fluctuations. However, if we observe the data over long time intervals, we can see trackable trends in the daily latency values. This may due to the fact that the traffic load of the cellular network fluctuates according to certain diurnal and weekly patterns [8]. Based on such an observation, we decompose the latency into two components: one that exhibits a trackable pattern over time and another that behaves like random noise, but possibly with certain autocorrelations.

To separate the two components, we apply a filter to the original data, aiming to average out the noise-like part. The latency obtained after filtering is the first component and the residual part is the second component. We considered three window sizes for the filter: 10, 100, and 1000. Fig. 3 depicts the autocorrelation for the raw data as well as the averaged traces. For the raw data, it always shows weak autocorrelation, which makes it hard to abstract the context information by time dependence. By applying a window filter, the autocorrelation increases visibly. It is clear that the filtered component is more trackable by the dependency of latency in the time domain. Obviously, the window size of the filter plays a major role in the first component's prediction accuracy. We will explore such impact in the simulations.



(a) Raw data (b) Filtered data with $W = 10$

(c) Filtered data with $W = 100$ (d) Filtered data with $W = 1000$

Fig. 3. Autocorrelation of raw and filtered data.

The structure of the proposed prediction framework is shown in Fig. 4. The first component, obtained after filtering, is called *baseline*. A time-series LSTM model is developed to predict the baseline after classifying the data samples according to the received signal feature. The second component is called *residual*, which approximately follows a t-distribution. The residual part is predicted using a hybrid statistical approach that combines several predictors. These predictors are based on sampling from the approximated pdf of measured residuals obtained by Epanechnikov Kernel functions. The final prediction of the residual part is a weighted sum of these predictors. The coefficients of the predictors can be adapted in an online fashion based on the instantaneous prediction error. Details of both components of the latency predictor are provided in the following two sections.

IV. LSTM-Based Prediction of Baseline Latency

In this section, we introduce the clustering-based LSTM design for predicting the baseline component. In contrast to the standard time series LSTM network, our proposed approach classifies the input data by signal features before training. By this way, our prediction approach have a prior expectation according to the signed label. Weak signal environments could also be detected by such a classifier. The classification is based on parameters related to access latency; specifically, RSRP, RSRQ, and SS. During the training phase, the input data are labeled and placed into different groups according to a combination of these access parameters. The LSTM networks are trained independently for various groups. For the testing phase, the input is classified based on the same common criteria, and then assigned to the corresponding trained LSTM network for prediction.

A. Data Classification

In a wireless environment, the received signal strength is a good indicator of channel conditions. It also impacts the response time between the transmitter and receiver. In our measured data, the average latency was observed to be lower

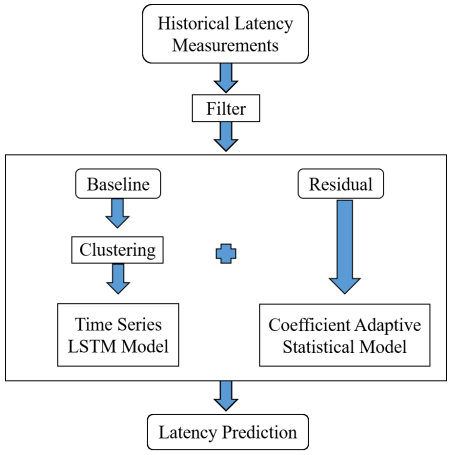


Fig. 4. Architecture of the proposed LSTM-integrated latency prediction framework.

when the signal quality is better. Furthermore, the signal strength fluctuates at a slower pace than the instantaneous latency. Based on this observation, before performing latency prediction, we classify the collected data into multiple groups according to the RSSI. We then develop a respective prediction training model for each group.

However, classifying data based on a single RSSI metric does not always provide good performance. Instead, we make use of the three available RSSIs for robust prediction performance, namely, SS, RSRP, and RSRQ. The SS is the transmission power of the serving LTE BS. RSRP is the average power over multiple LTE resource elements that carry cell-specific reference signals within the considered measurement frequency band. RSRQ is a measure of the signal quality, defined as the RSRP per OFDM resource block, which is used when RSRP is not sufficient to make a reliable handover decision. To classify data based on SS, RSRP, and RSRQ, we apply a clustering method called k-medoids [13]. Essentially, k-medoids is a distance-based unsupervised learning algorithm that clusters the input data into K groups with different labels. The k-medoids algorithm is described as follows:

1) In the three-dimensional sample space $\mathcal{S}$, randomly generate K medoids m, $m \in \mathcal{M}$;
2) Calculate the Euclidean distances between all the data points in $\mathcal{S}$ and medoids in $\mathcal{M}$. Then, assign each data point to the cluster with the minimum distance;
3) In each cluster, test each data point as a potential medoid m by examining if the average distance is reduced;
4) If so, every data point in $\mathcal{S}$ is then assigned to a cluster with the updated m;
5) Repeat steps 2 through 4 until the minimum average distance is achieved.

Applying the k-medoids algorithm, the original data set is classified into several clusters according to the values of SS, RSRP, and RSRQ. Naive Bayes filter and Support vector machine can also solve such a classification problem. However, these methods rely on splitting the problem into k sub-binary classifications, with exponentially increasing complexity but

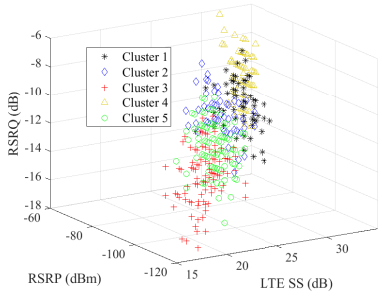


Fig. 5. Clustering measured latencies based on different RSSIs.

with comparable prediction accuracy to k-medoids [12]. As shown in the example in Fig. 5, the clusters with better signal quality are more likely to achieve lower latency.

B. Architecture of the LSTM Network

Our measurements reveal that the latency at a given time is impacted by both short-term as well as long-term historical information, and the latency follows a trackable pattern over time once the small-scale variations are filtered out. This motivates us to employ the LSTM algorithm for our prediction problem. LSTM is an advanced recurrent neural network (RNN) architecture that can intelligently combine long-term and short-term information. It stores the memory in the hidden layers and controls the information flow (i.e., determines what to remember and what to forget) by adapting the parameters of several gates. In each hidden layer, there is a forgetting function that decides the weight of the latest input. LSTM is also computationally efficient compared to other neural network architectures, such as convolutional neural networks (CNNs) and multi-layer perceptron (MLP). In our case, we can achieve a prediction accuracy of less than 10% of the RMSE with only 20 hidden units.

To train an LSTM network, all data should be normalized. The collected data points are divided into two parts: one for model training (90% of data) and one for testing (10% of data). The number of features is set to one, as the model only depends on historical latency data to make its prediction. For the time series prediction, first several samples contributes less for trends capture. Such that the initial learning rate is set to 0.005 and the learning rate drop factor is set to 0.2. The measured ith latency is denoted as X_i, and the total number of measurements is n. The classification function $g(\cdot)$ can be expressed as: $g(x) = x_k$, if x belongs to cluster k, $k = 1, 2, ..., K$

After clustering, we divide the input into K groups: X_1, X_2,..., X_K. For each group X_k, the predicted ith latency can be represented as $Y_{k,i}$:

$$Y_{k,i} = f(g(X_k)) = f(X_{k,i-1}, X_{k,i-2}, ..., X_{k,1}) \quad (1)$$

where $f(\cdot)$ is a mapping between the measured latency values up to $X_{k,i-1}$ and the predicted value. This mapping is determined by the parameters and setting of the LSTM network. A typical LSTM network consists of multiple LSTM cells that determine the parameters of the hidden layers. The

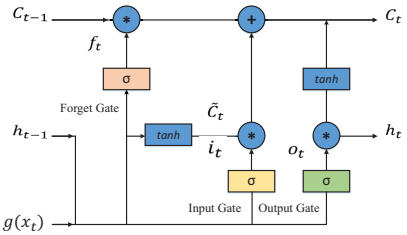


Fig. 6. Architecture of an LSTM cell used in this paper.

architecture of a cell is shown in Fig. 6. At each time step t, $g(x_t)$ is the classified system input and h_t is the output of the LSTM cell. The cell output at the previous time step, h_{t-1}, is combined with the current system input $g(x_t)$ to form the input for the current cell. The state of each cell is C_t, which records the system memory. C_t is updated at each time step. To control the information flow through the cell, several gates are applied, including an input gate (i_t), output gate (o_t), and forget gate (f_t). Each gate generates an output between 0 and 1, where the value of the output is calculated by a sigmoid (σ) function. An output of 0 indicates that the input of the gate is totally blocked while an output of 1 indicates all information of the input is kept in the cell. The input, output, and forget gates are calculated as follows:

$$\begin{aligned} i_t &= \sigma(W_i g(x_t) + U_i h_{t-1} + b_i) \\ o_t &= \sigma(W_o g(x_t) + U_o h_{t-1} + b_o) \\ f_t &= \sigma(W_f g(x_t) + U_f h_{t-1} + b_f) \end{aligned} \quad (2)$$

where W_i, W_o, and W_f are the weights of the three gates; U_i, U_o, and U_f are the corresponding recurrent weights; and b_i, b_o, and b_f are the bias values of the three gates.

Similar to the gate function, we combine the current inputs and previous cell state C_{t-1} to update the cell state. The difference is that instead of a sigmoid, the inputs will be processed by a hyperbolic tangent function that generates an output between -1 and 1:

$$\widetilde{C}_t = \tanh(W_c \cdot g(x_t) + U_c \cdot h_{t-1} + b_c) \quad (3)$$

After the update, $\widetilde{C}_t$ is multiplied by the output of the input gate, which is then used as the first component to update the cell state. Another component for updating the cell state is the previous cell state, which is processed by the forget gate to determine how past data is to be utilized. With the two components, the cell state at time t is updated as:

$$C_t = f_t C_{t-1} + i_t \widetilde{C}_t \quad (4)$$

The output of the cell h_t, which will be used at time $t+1$, is calculated by the multiplication of the output gate and the tanh function of the current cell state:

$$h_t = o_t \tanh(c_t) \quad (5)$$

C. LSTM Network Training and Testing

The training procedure is performed offline while the prediction is performed online with the trained LSTM networks. Based on the outcome of K clusters of the k-medoid

algorithm, we set K independent LSTM networks. For each network, we first normalize the input data as follows:

$$X_i{}' = \frac{X_i - \bar{X}_i}{\sum_{i=1}^{n} \frac{1}{n}(X_i - \bar{X}_i)^2}, \text{ for } i = 1, 2, ..., n \quad (6)$$

where $\bar{X}_i$ is the sample mean latency of the set.

Each LSTM network is trained to find the optimal regression model between the input and output sequences. The number of cells is set to 20. Cell features will periodically flow into the next layer to evaluate the system state. In each training iteration, the weights of the input and output are adapted by multiplying the cell state matrix, and the feedback of error is then used to update the cell states and the parameters of the corresponding LSTM network. The training is completed when RMSE reaches the target threshold.

V. Prediction of the Residual Component

A. Predictor Design

To capture the impacts of long- and short-term historical latencies, we propose to use a combination of three predictors for the residual that remains after subtracting the baseline component.

1) PDF Approximation: From Fig. 2, we observe that the histograms of cloud and edge node latencies can be approximated by a combination of two normal distributions. By fitting the data, we can perform sampling according to these approximate PDFs and use the outcome for prediction. However, the exact distributions are unknown and cannot be expressed in closed form. Thus, we investigate and evaluate several approaches that aim at finding a good approximation for the latency PDF. Specifically, the probability that the latency equals to x is calculated using the following Epanechnikov Kernel function:

$$\widehat{f}_{w_s}(x) = \frac{1}{n}\sum_{i=1}^{n} K_{w_s}(x - x_i) = \frac{1}{nw_s}\sum_{i=1}^{n} K\left(\frac{x - x_i}{w_s}\right) \quad (7)$$

Recall that n is the total number of samples, x_i is the ith data point, and K_{w_s} is a non-negative Epanechnikov Kernel function with window size $w_s > 0$. After applying the Epanechnikov Kernel function, the distribution of the original data can be approximated by a PDF of a continuous variable.

A predicted latency value corresponding to a higher $\widehat{f}_{h_k}(x)$ will be selected with higher probability during sampling. This way, the probabilistic sampling is a repetition of the actual process that generates latency values. Considering that the time dependence is stronger with shorter sample distance, we introduce two predictors: short-term sampling (STS) and long-term sampling (LTS). STS is using the most recent latency distribution as sample space, whereas LTS can make full use of all the collected latency. The prediction value is then represented as Y_{STS} and Y_{LTS}.

2) Moving Average: To make use of the correlation in latency over time and to better capture the instantaneous latency pattern, we augment the previous two predictors with moving average (MA) predictor, given by the weighted sum of latency values over a sliding window. In this paper, we employ an exponentially weighted moving average (EWMA) approach over the given window size. We evaluate the prediction performance under different window sizes in our experiments. The predicted latency can be expressed as:

$$Y_i = Y_{i-1} + \alpha(X_i - Y_{i-1}) \quad (8)$$

where α is a coefficient that can be expressed as $\alpha = \frac{2}{w_m+1}$ and w_m is the window size of EWMA function. The predicted value by moving average function is denoted as Y_{MA}

B. Combining Multiple Predictors

From the collected data, it can be observed that during the periods when the latency changes rapidly, recent latencies have a more significant impact on the current latency compared to latencies taken several hours before. During the online prediction process, the weights assigned to the predictors can be adjusted according to the prediction error. This way, we can dynamically optimize the relative importance of short- and long-term information. In addition, as a key system parameter, the size of the sliding window should be evaluated to optimize the prediction performance.

Let's consider combine Y_{STS}, Y_{LTS}, and Y_{MA} to make full use of each part's merit. Let a, b, and c be the weights (coefficients) of the three predictors, where $a + b + c = 1$. Then, the final prediction outcome is given by:

$$Y_i = aY_{\mathrm{STS}} + bY_{\mathrm{LTS}} + cY_{\mathrm{MA}}. \quad (9)$$

C. Coefficients Adaption

To get the best forecasting results, the coefficients a, b, and c in equation (9) are updated using a Kalman filter. Let a_i, b_i, and c_i denote the values of these coefficients at the ith step. We define the error matrix E_i as $Y_i - X_i$. E_i is used to record the prediction error and provide feedback for the prediction in the next step. Consider j sets of coefficients, $\{a_1, b_1, c_1\}, ..., \{a_j, b_j, c_j\}$. Y_i can be written as:

$$Y_i = \begin{bmatrix} Y_{\mathrm{STS}} & Y_{\mathrm{LTS}} & Y_{\mathrm{MA}} \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & ... & a_{j-1} & a_j \\ b_1 & b_2 & ... & b_{j-1} & b_j \\ c_1 & c_2 & ... & c_{j-1} & c_j \end{bmatrix}. \quad (10)$$

After observing the actual latency X_i, the controller calculates $E_i = \begin{bmatrix} E_1 & E_2 & ..... & E_{j-1} & E_j \end{bmatrix}$ and determines $j' \in \{1, 2, ..., j\}$ that minimizes E_j:

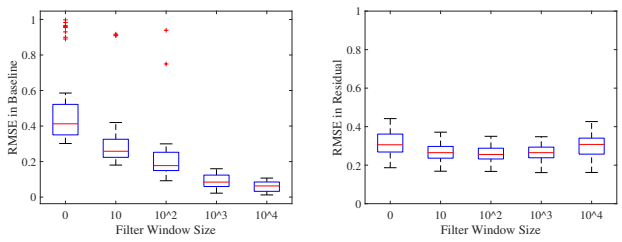
$$j' = \arg \min_{j' \in \{1,2,3,...,j\}} E_j \quad (11)$$

The coefficients $\{a_{j'}, b_{j'}, c_{j'}\}$ is used for the next step prediction.

VI. Performance Evaluation

A. Impact of Filter Settings

We first evaluate the impact of the window size that is used to split the data between baseline and residual. The RMSE performance of the baseline part under different window sizes is shown in Fig. 7(a). We can see that the RMSE decreases



(a) RMSE of baseline vs. window size of filter (b) RMSE of residual vs. window size of filter

Fig. 7. RMSE performance under different window sizes.

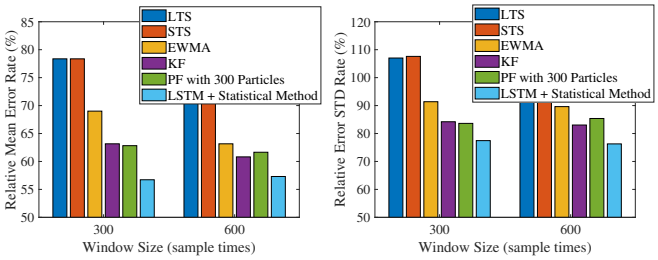
quickly with filter window size, and then stabilizes. This is because the rapidly varying component is removed from the original data, making the baseline part more trackable. Besides, the RMSE tends to be concentrated around its mean as the window size increases, indicating that the prediction performance is becoming more stable. The impact of the window size on the prediction error for the residual part is plotted in Fig. 7(b). To obtain more accurate latency estimates without significantly increasing complexity, we choose a window size of 1000 in the remaining experiments, which achieves an RMSE of less than 10%.

By separating the 'noisy" component from desired trends, we remove the interference of irrelevance. Such that the training process of the designed LSTM network requires less computational resources. As introduced in parameter setting, cell matrix and layer number are downsized, meanwhile, error rate can still be controlled into the desired range.

B. Evaluation of The Proposed Prediction Approach

In the performance evaluation, prediction methods are compared by the measurement data reserved for testing. Measured latencies are collected at fixed location and during driving as described previously. We define the prediction error (e) as the difference between predicted value and actual latency. Both the mean error ($\bar{e}$) and error STD (θ) are reported. We compare all the methods with a sample-mean predictor. The ratio of the mean error and the error STD between tested methods and the sample-mean predictor are denoted ϵ and μ, respectively.

Fig. 8 compares different methods in terms of their relative prediction error and STD of this error. The LTS and STS methods decrease ϵ by 20%, which indicates a weak performance if we depends only on sampling predictors. The classical approaches for comparison, including Kalman Filter (KF) and Partial Filter (PF), can achieve a relatively low prediction mean error. However, μ value is around 85% for both of them, showing that these approaches are not guaranteed to perform well in all scenarios. The proposed approach achieves the best performance in both relative mean error and relative error STD and can reduce the mean error by 45%, which achieve around half of the benchmark. Besides, the error STD is reduced by 25%, showing that our approach can achieve the best performance in terms of both prediction error and stability among all methods.



(a) Relative mean error (ϵ) comparison (b) Relative error STD (μ) comparison

Fig. 8. ϵ and μ comparison between different prediction methods using collected measurements

VII. Conclusion

In this paper, we proposed an LSTM-integrated latency prediction framework for MEC supported delay-sensitive V2X applications. We first perform latency measurements from both fixed location and mobility scenarios in an LTE-based MEC network. Based on the observation of collected latency data, we develop a prediction model based on a combination of LSTM and statistical approaches. Performance based on real-time collected data shows that both the integrated LSTM can achieve lower the prediction error with more stability.

References

[1] NGMN Alliance, "V2X white paper," Jun. 2018.
[2] V. John et al., "Estimation of steering angle and collision avoidance for automated driving using deep mixture of experts," *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 4, pp. 571-584, Dec. 2018.
[3] 5GAA, "Toward fully connected vehicles: Edge computing for advanced automotive communications," White Paper, Dec. 2017.
[4] Y. Xiao, M. Krunz, H. Volos, and T. Bando, "Driving in the fog: Latency measurement, modeling, and optimization of LTE-based fog computing for smart vehicles," in *Proc. IEEE SECON'19,* Boston, MA, June 2019, pp. 1–9.
[5] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Trans. Veh. Technol.,* vol. 68, no. 5, pp. 5031–5044, May 2019.
[6] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.,* vol. 6, no. 3, pp. 4005–4018, June 2019.
[7] 3GPP, "LTE; requirements for Evolved UTRA (E-UTRA) and Evolved UTRAN (E-UTRAN)," 3GPP TR 25.913, Feb. 2010.
[8] H. Volos, T. Bando and K. Konishi, "Latency modeling for mobile edge computing using LTE measurements," *IEEE VTC'18*, Chicago, IL, pp. 1–5.
[9] Y. Wang, Y. Shen, S. Mao, X. Chen, and H. Zou, "LASSO & LSTM integrated temporal model for short-term solar intensity forecasting," *IEEE Internet Things J.*, vol.6, no.2, pp.2933–2944, Apr. 2019.
[10] X. Wang, Z. Yu, and S. Mao, "Indoor localization using magnetic and light sensors with smartphones: A deep LSTM approach," *Mobile Networks and Applications (MONET) Journal*, DOI: 10.1007/s11036-019-01302-x.
[11] I. Burago, M. Levorato, and A. Chowdhery, "Bandwidth-aware data filtering in edge-assisted wireless sensor systems." in *Proc. IEEE SECON'17* pp. 1–9. June 2017.
[12] C. Kyrkou, C. Bouganis, T. Theocharides and M. M. Polycarpou, "Embedded Hardware-Efficient Real-Time Classification With Cascade Support Vector Machines," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 1, pp. 99-112, Jan. 2016.
[13] E. Schubert and P. J. Rousseeuw. "Faster k-Medoids clustering: improving the PAM, CLARA, and CLARANS algorithms." in *Proc. International Conference on Similarity Search and Applications*. Springer, Cham, pp. 171-187, 2019.