Latency Estimation and Computational Task Offloading in Vehicular Mobile Edge Computing Applications

Wenhan Zhang, *Student Member, IEEE*, Mingjie Feng, *Student Member, IEEE*, Marwan Krunz, *Fellow, IEEE*

***Abstract*—Mobile edge computing (MEC) is a key enabler of time-critical vehicle-to-everything (V2X) applications. Under MEC, a vehicle has the option to offload computationally intensive tasks to a nearby edge server or to a remote cloud server. Determining where to execute a task necessitates accurate estimation of the end-to-end (E2E) offloading delay. In this paper, we first conduct extensive measurements of the round-trip time (RTT) between a vehicular user and edge/cloud servers. Using these measurements, we present a latency-estimation framework for optimal task offloading. The propagation delay, measured by the RTT, is divided into two components: one that follows a trackable trend (baseline) and the other (residual) that is quasi-random. For the baseline component, we first cluster measured RTTs into several groups, depending on signal strength indicators. For each group, we develop a Long Short-Term Memory (LSTM) regression model. A statistical approach is provided for predicting the residual component, which combines the Epanechnikov Kernel and moving average functions. Predicted propagation delays are incorporated into virtual simulations to estimate the transmission, queuing, and processing delays, hence accounting for the E2E delay. Based on the estimated E2E delay, we design a task offloading scheme that minimizes the offloading latency while maintaining a low packet loss rate. Simulation results show that the proposed offloading strategy can reduce the E2E delay by approximately 60% compared to a random offloading scheme while keeping the packet loss rate below 3%.**

***Index Terms*—V2X applications, mobile edge computing, task offloading, latency prediction, LSTM, E2E delay.**

I. Introduction

Intelligent transportation systems (ITS), including connected and autonomous vehicles (CAV), have recently been at the forefront of research in both academia and industry. 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) [2]. 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 involve executing computationally intensive tasks in near-real-time. For example, object detection (e.g., of pedestrians, bikes, vehicles, etc.) often involves extensive, real-time processing by deep learning [3], [4]. However, such processing would be prohibitive for the in-vehicle embedded processor [5]. Instead, computationally intensive V2X tasks may be offloaded to a remote cloud server with sufficient computing resources. However, the end-to-end (E2E) communication latency between the vehicle and the remote server can be excessively high, as the connection traverses multiple hops with varying network dynamics and states of congestion. As an alternative, the task may be offloaded to a nearby edge server [6], [7]. Exploiting edge servers close to base stations (BS) or roadside units (RSUs) can significantly reduce the communication latency. However, such servers are unlikely to have the same computational capacity of a full-fledged data center. To minimize the E2E latency, the offloading decision should take into consideration the servers' availability, computing capacities, and task attributes.

Determining where to execute a V2X task necessitates accurately predicting its communication and computing latencies. While the computing latency can be estimated based on the task size and processing power, estimating the communication latency is more challenging due to the fluctuating conditions of the network path. In most existing works, the communication latency is approximated by the transmission time [8], [9], obtained from the task size and link rate. Such an approximation overlooks the channel's *access latency* between a connected vehicle and an edge/cloud server. As demonstrated in our measurements, the access delay can be pretty significant and is highly dynamic. Unlike the transmission latency, which can be estimated a priori, there is no apparent relationship between the access latency and the task size. A measurement-based forecasting approach would be more appropriate for estimating such latency. After a packet arrives at an edge or cloud server, it is queued before processing, so the load of the selected server clearly impacts the queueing time. Moreover,

Manuscript received January 2, 2023; revised June 22, 2023 and September 25, 2023; accepted November 5, 2023. This work was supported in part by NSF (grants CNS-2229386, CNS-1910348, CNS-1813401, and IIP-1822071) and by the Broadband Wireless Access & Applications Center. Any opinions, findings, conclusions, or recommendations expressed in this paper do not necessarily reflect the views of NSF.An abridged version of this paper appeared in proceeding of the GLOBECOM'20 Conference, Dec. 2020 [1]. The review of this paper was coordinated by Prof. Hung-Yun Hsieh. (Corresponding author: Wenhan Zhang.)

W. Zhang and M. Krunz are with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, 85721 USA. M. Feng is with Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, 430074 China. He was with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ, 85721 USA. Email: wenhanzhang@arizona.edu, mzf0022@auburn.edu, krunz@arizona.edu.

Digital Object Identifier 10.1109/TVT.2023.XXXXXXX

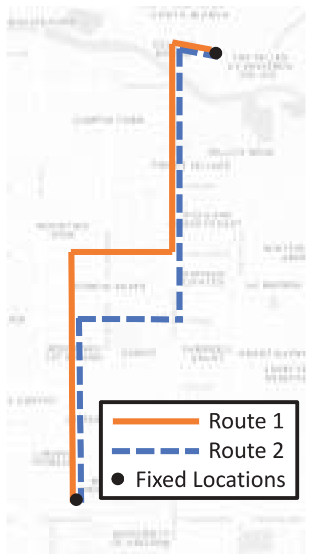
the processing latency at a given server usually varies and affects the queueing time. We study both the queueing and the processing latencies to determine the overall computing latency.

The differences between edge and cloud servers in computing capacity, network latency, storage limitation, and other features accentuate the importance of the offloading strategy and its impact on reducing the E2E latency. Previous research on this topic (e.g., [10]–[12]) neglects that packets of offloaded tasks may be queued at the server. Given that an edge server has much fewer computational and storage resources than a cloud server, packets received by an edge server may end up being discarded due to buffer overflow. In reality, due to traffic dynamics, the load of a server can be higher than its computational capacity, so analyzing the queueing process is essential for evaluating the offloading performance. Our latency analysis considers communication and computing delays (i.e., queueing and processing) associated with the offloading decision. The proposed offloading scheme reduces the E2E delay while maintaining a low packet loss rate.

In this paper, we first propose a prediction framework for the round-trip times (RTTs), which combines Long Short-Term Memory (LSTM) with statistical regression approaches. Our techniques rely on extensive measurements of the RTTs of 'ping' messages sent from an in-vehicle mobile phone to either an edge server or a data center. Preliminary modeling and analysis of the latency measurements were performed in [1]. In the underlying paper, we expand on such analysis by reporting the RTT statistics for different fixed locations, driving routes, and times of day. In addition, we analyze the correlations between the latency and signal-strength indicators, and cluster the measured data points using the K-medoids approach. By projecting the clustered data from a high-dimensional space onto 2D space using distributed stochastic neighbor embedding (t-SNE), we show that the data points are separable based on signal strength features.

We combine the predicted RTT with transmission, processing, and queueing delays for each task to come up with an estimate of the end-to-end (E2E) delay. Our model incorporates the task size, task complexity, and the transmission/processing rates of cloud/edge servers. Based on this model, we study the success rate and task latency under various settings. Finally, we propose a task offloading scheme based on the estimated E2E delay. The proposed scheme enables vehicular users to dynamically select between a cloud or edge server so as to reduce the overall E2E delay. We compare the performance of the proposed offloading scheme with three other offloading schemes. The main contributions of this paper are summarized as follows:

- We use a customized smartphone app called *Delay Explorer* to collect thousands of traces of the access delay (the "RTT") between an in-vehicle mobile device and an edge/cloud server. These traces were collected at two different fixed locations and over two driving routes. We provide comprehensive statistical analysis of the collected data and highlight important trends.
- By applying an appropriate filter, we split the captured data into two parts: A trackable *baseline* part and a *residual* part. Mechanisms are provided for online estimation of each part. For the baseline part, the data is first clustered into groups according to received signal indicators. Then, an LSTM network is trained and used to predict future baseline values of each cluster. For the residual part, we propose a hybrid statistical prediction approach that combines Epanechnikov Kernel function-based probabilistic sampling and moving average function-based prediction.
- The proposed latency prediction method is compared with five other prediction methods. The results indicate 45% reduction in the prediction error compared with a sample-mean predictor. Relative to the Kalman filter-based and particle filter-based predictors, our approach is shown to reduce the prediction error by around 15%.
- We augment our communication-latency prediction model with predictions of the queuing and processing delays so as to estimate the E2E delay of a given task. Based on the estimated E2E delay, we propose an adaptive task offloading scheme that aims at minimizing the total delay. The proposed offloading scheme is shown to reduce the latency by 60% compared with a random server-selection method while maintaining the packet loss rate (due to buffer overflow) below 3%.



(a) (b)

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

II. Related Work

In an MEC system, a user may choose to be served by a cloud node[1] (CN) or an edge node (EN). The latency of edge systems has been modeled and analyzed in various previous works. The authors in [13] aimed to reduce the user-experienced delay through prediction, assuming a fixed end-to-end propagation delay. In [14], the authors estimated the network latency by solving a Matrix Completion problem. The authors in [15] estimated network latency using partial

[1]in this paper, we use 'cloud node' to refer to a server located in a remote data center.

measurements, thereby reducing the computational complexity. However, these works assumed that the latency depends only on a fixed set of parameters (e.g., transmission rate, distance between nodes, and packet size) and overlooked the real dynamics of the network. Our measurements show that even for packets with the same parameters, the latency varies constantly, and the network dynamics can significantly impact the delay experienced by users. In contrast to the above papers, we consider the uncertainty in communication and computing latencies and propose an LSTM-based model to capture the latency dynamics. Although LSTM models have been applied in several time-series prediction problems (e.g., [16], [17]), their application in V2X latency prediction is not straightforward. In our work, an LSTM-based frame is used to learn the temporal dependency in observed latencies and intelligently combine long- and short-term information. Therefore, it is expected to improve prediction accuracy compared to time-invariant models such as those in [15] and [18].

Several previous works investigated offloading schemes for MEC systems and attempted to optimize the latency experienced by the user. In [20], the authors developed a fault-tolerance methodology for latency-aware edge computing applications based on checkpointing and replication. In [21], the author formulated a non-convex mixed-integer nonlinear programming problem with the objective of minimizing energy consumption under delay constraints. Finally, in [22], the authors proposed joint incentive design and resource allocation for edge computing based on Lyapunov optimization. However, the authors in [20]–[22] mainly focused on minimizing the task execution and transmission delay, oversimplifying the queueing delay at the edge server, which did not fully analyze and optimize the E2E delay. In addition, the authors in [19]–[22] rely on the underlying assumption that the various components of E2E latency can be accurately estimated, while the interaction between latency estimation and task offloading was overlooked. In our work, we use the estimated E2E delay to design an adaptive task offloading strategy. The proposed strategy enables vehicular users to dynamically select between the mobile cloud and edge server so as to reduce the overall E2E delay.

III. Latency Measurements

To obtain real measurements of the communication latency between a vehicle and edge/cloud servers, we use a customized smartphone application called *Delay Explorer*. This app sends a stream of ping packets to the IP addresses of the edge and cloud nodes, and records the feedback from these IPs. The interval between two consecutive ping packets is set to 500 ms, and these packets are sent via an AT&T LTE network. Ping packets are usually small in size, and their transmission and processing times are neglectable compared to normal V2X packets. Furthermore, when the destination IP address receives a ping packet, it will immediately send an acknowledgement back to the user. Therefore, the RTTs of these ping packets can be used to estimate the RTT between a user and a server in delay-sensitive applications. Because edge servers are typically located close to mobile users, the IP address of the first node that responds to the ping message can be regarded as the location of the edge server. In our measurements, the cloud node is an Amazon Web Server (AWS), located at the Amazon cloud service center. *Delay Explorer* also records other useful parameters besides RTT, including received signal indicators, velocity, GPS information, and others. In our prediction framework, we use the LTE Signal Strength (SS), the Reference Signal Received Power (RSRP), and the Reference Signal Received Quality (RSRQ) to classify a V2X message and predict its latency. These metrics will be discussed later in Section V. Fig. 1(a) shows the interface of *Delay Explorer*.

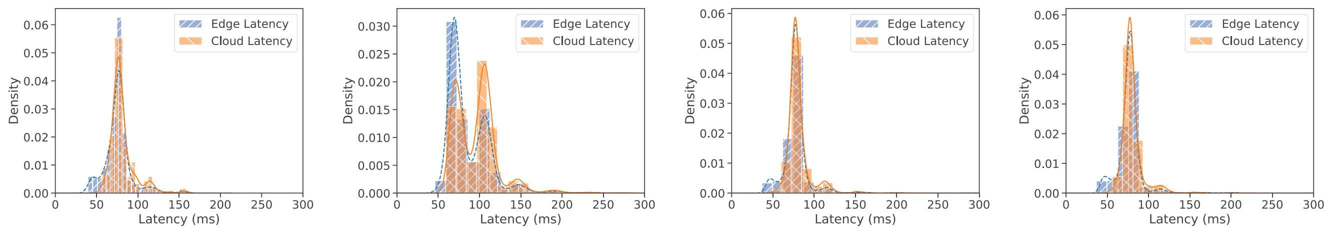
TABLE I
Statistics of Measured Latency (in msec) for Edge and Cloud Nodes

Types	Statistics	L1	L2	Ave.	R1	R2	Ave.
Edge Latency	Mean	75.66	86.45	81.05	70.49	72.94	71.71
	STD	15.85	26.15	21.00	17.48	17.51	17.49
	STD/Mean (%)	20.95	30.25	25.91	24.80	24.01	24.39
	Median	76.73	75.13	75.93	72.22	72.17	72.19
	Conf. 90%	90.02	113.23	101.62	86.72	86.19	86.45
Cloud Latency	Mean	80.92	96.41	88.66	81.36	80.49	80.92
	STD	14.68	26.41	20.54	16.4	15.28	15.84
	STD/Mean (%)	18.14	27.39	23.17	20.16	18.98	19.57
	Median	77.45	99.31	88.38	77.82	77.81	77.81
	Conf. 90%	97.09	118.85	107.97	94.31	90.1	92.21

We collected more than 1,600,000 RTT measurements both at fixed locations and while driving. The fixed-location scenarios include an office and an apartment (see Fig. 1(b)). For the mobile scenarios, latency measurements were taken while driving along the routes in Fig. 1(b). These measurements represent realistic vehicular conditions, including node and traffic densities, as well as channel conditions that vary along the route. In addition, to provide a good representation of practical use cases, data collection was conducted at different time periods, including weekdays, weekends, mornings, afternoons, etc.. The measurements are divided into training and testing parts, with the latter part used to evaluate the performance of the proposed predictors. Fig. 2 shows the distributions and key statistics of measured RTTs. As expected, the edge server has a lower mean RTT than the cloud server, but it also has a higher standard deviation (STD).

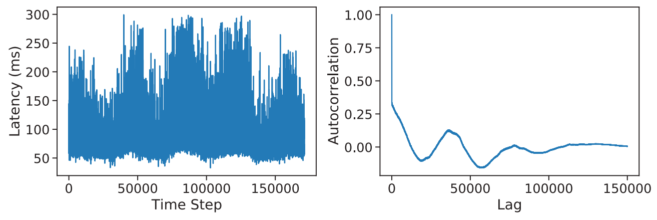
A. Impact of Locations and Routes

Table I depicts the mean, STD, and median of the RTT for two fixed locations (L1, representing an office on campus, and L2 representing an apartment), as well as two major driving routes (R1 and R2, as shown in Fig. 1(b)). It can be observed that the RTTs of different network nodes can vary significantly in some locations/driving routes. However, the mean and median values of latency for the edge node are always lower than those of the cloud node. In addition, the edge node usually has a higher STD than the cloud node. To evaluate the latency tolerance, we define *confidence* as the latency value where a particular fraction of measured latencies is below. For example, a 90% confidence value for L1 edge latency is 90.02 ms, which means 90% of the measured edge latencies for L1 are less than 90.02 ms. We observe that the confidence for mobile cases is less than



(a) Fixed location 1 (b) Fixed location 2 (c) Route 1 (d) Route 2

Fig. 2. Probability density distribution and kernel density estimation comparison between the cloud-node latency and edge-node latency at different locations and during different routes.



(a) The time-series trends (b) Auto-correlation

Fig. 3. Continuous measurements over five weekdays (Monday to Friday) at the fixed location 2.

for fixed-location cases. This is because the measurements during driving were taken along with the main roads for urban areas; however, the fix-location measurements were always in indoor environments (i.e., campus and apartment). We plot the probability density distribution for L1, L2, R1, and R2 in Fig. 2. The PDFs vary significantly for different locations, as shown in Fig. 2(a) and (b), which may be attributed to the difference in the base station configurations at these locations. In contrast, the PDFs of measured latencies along the routes in Fig. 2(c) and (d) look similar, which may be justified by the fact that the two paths are close to each other.

To explain the observed latency fluctuations, we normalized the STD by the corresponding mean latency at different locations and routes, and added the resulting metric in Table I. For EN latency, L1 has the lowest normalized STD of 20.95%, which is lower than R1 (24.80%) and R2 (24.01%). L2 has the highest normalized STD of 30.25%. A similar relationship is observed for the cloud latencies. In other words, the normalized STD of the latency at fixed location L1 is lower than its mobile counterpart for both EN and CN. The higher fluctuations for L2 are attributed to two peaks in the histogram.

B. Peaks and Periodicity

For scenarios L1, R1, and R2, the histogram for the RTT depicts one peak and one bulge at around 70 ms and 110 ms, respectively. In contrast, for L2 we observe two peaks at 67 ms and 106 ms, and two bulges at 144 ms and 190 ms. The gap between successive peaks/bulges is around 40 ms, which corresponds to the periodicity of the Scheduling Request (SR) and Hybrid ARQ (HARQ) messages in LTE systems [7], [24]. Specifically, in a synchronized LTE system, radio resources are divided into frames of 10 ms each. The resource block for a certain function is usually assigned to a subframe within the frame. If the UE does not receive an acknowledgment (ACK) from the base station (BS), it has to wait and resend the request in the next period. The 40 msec periodicity due to SR messages and HARQ retransmissions was also observed by other researchers (see, for example, the measurements in [7], [24]).

The two peaks in Fig. 2(b) are caused by the fact that the measurements in scenario L2 were taken over a time span that includes late afternoons, evenings, night hours, and mornings. During night hours, the traffic load is generally light, so the BS may switch to sleep mode to reduce energy consumption by deactivating certain units [25], [26]. If the UE sends its request during such times, the BS may fail to respond on time, and the UE has to resend such a request in the next period. Accordingly, in the case of scenario L2, a relatively high density of requests will end up with about 40 ms of additional delay, creating a second peak at 106 ms.

We also observed that the heights of the peaks differ for EN and CN. In our measurement setting, we assume the first-hop node is an EN. When the ping request arrives at the EN, the response is sent from the BS immediately. However, when we ping the CN, the BS has to wait for the response from the cloud before sending feedback to the UE. This results in a higher latency for the CN. Therefore in Fig. 2(b), there is less density around the first peak in the case of cloud latency than in the case of the edge latency in Fig. 2(b).

C. Impact of Time Period

To better show the statistics and patterns of latency, we plot in Fig. 3 the latency during five continuous weekdays. The figure indicates that the average value of latency varies over time during the day. This may be caused by the changes in the traffic of user devices or the signal power. We also plot the autocorrelation of the weekly measurements in Fig. 3(b), where a periodical pattern is seen over the days. The autocorrelation drops quickly to around 0.3 as the lag increases to larger than 1. It then slowly decreases to a negative value after about 20,000 points (about 12 hours) and then starts to increase. It achieves another peak at about 40,000 points (about 24 hours), then drops again periodically. Note that when the lag is 40,000, the latency in Fig. 3(a) also shows a high expected value. Such peaks and dips intersect with periodical lags, indicating that the latency is correlated with the time of the day. The autocorrelation diminishes to zero when the lag is large. The observed periods indicate the trends behind the raw data, and we can use them to estimate the actual latency over time.

IV. Overview of Prediction Framework

Two key observations can be made based on our analysis of the latency measurements. First, the measurements show significant fluctuations that closely follow a Gaussian distribution or a combination of multiple Gaussian distributions. Second, when viewed over long intervals, the data depict daily trends. These two features were also observed in [28]. Based on these observations, we propose to filter the data first and use an LSTM network to learn the temporal dependencies of such trends. Given that the latency is non-stationary and its distribution is not in closed form, we apply weighted sum of statistical predictors to model the residual part of the measured latencies.

Fig. 4(a) depicts the autocorrelation of the raw data. It can be seen that the correlation of the raw data is not significant. Therefore, we apply moving average-based filter functions to process the data, which facilitates the extraction of correlation. Fig. 4(b)–(d) depicts the autocorrelation of the traces obtained after applying filters with three window sizes: 10, 100, and 1000. We observe that the temporal dependencies increase by applying these filters. In other words, the filtered component becomes more trackable in the time domain. Therefore, we apply a filter function and decompose the latency into two components (see Fig. 5): one that exhibits a trackable pattern over time (which we call a *baseline*) and another that behaves like correlated random noise, i.e., colored noise, which we call *residuals*.

A time-series LSTM model is designed to predict the baseline. Even with an LSTM prediction model, the accuracy is still limited. Accordingly, we exploit the dependency between received signal indicators and access latency to cluster the data and apply a per-cluster LSTM predictor. We show that such clustering strategy results in improving the prediction accuracy in Section V. The second component, the residuals, is predicted using a hybrid statistical approach that combines several predictors. These predictors are based on sampling the approximated pdf of measured residuals obtained by Epanechnikov Kernel functions. The final prediction of the residual is a weighted sum of these predictors, and the coefficients can be adapted in an online fashion according to the instantaneous prediction error.

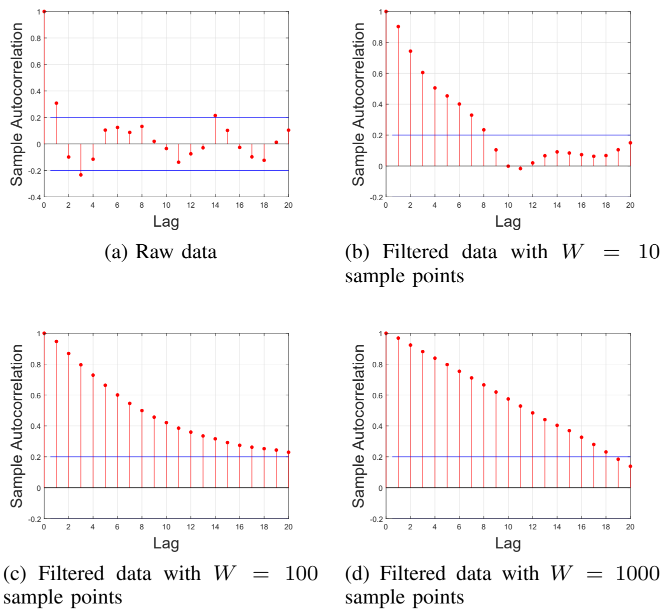


Fig. 4. Autocorrelation of raw and filtered data (baseline component) for a representative trace.

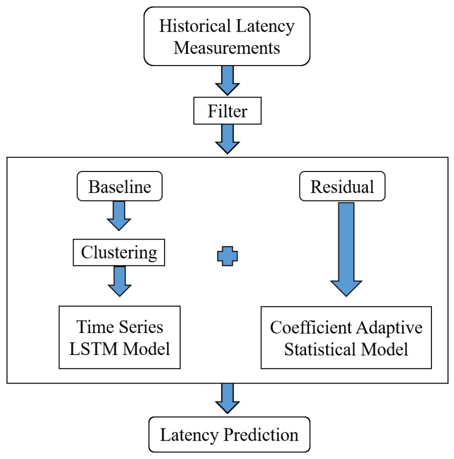


Fig. 5. Architecture of the proposed LSTM-integrated latency prediction framework.

V. LSTM-Based Prediction of Baseline Latency

To predict the baseline component, we design a clustering-based LSTM network that utilizes the received signal indicators to improve prediction accuracy. In a wireless environment, the received signal strength is a good indicator of channel conditions. We first depict the correlations between the latency and other measured parameters in Fig. 6. In contrast to signal-strength-related metrics that show strong correlations with latency, the impact of the vehicle's speed on latency is negligible, i.e., the correlation coefficient is quite low. Thus, we focus on using the signal strength indicator to assist the latency estimation. The signal strength depends more on the receiver's environment (e.g., reflections, diffraction, and scattering from buildings and blocks) and fluctuates at a slower pace. Therefore, we adopt three signal strength indicators as the criteria for data clustering before performing latency prediction. K-medoids algorithm is used to group the collected data. After data clustering, multiple LSTM networks are trained independently for different groups. During the testing phase, the input is clustered based on the same criteria, and then assigned to the corresponding trained LSTM network for prediction.

A. Correlation Between Latency and Signal Strength Indicators

We use RSRP as an example to show that the strength of the received signal is correlated to latency. In Fig. 7, we plot the measured latency vs. RSRP for a segment of dataset. We can see that the latency values mainly fall in the range between 65 and 85 ms, and they roughly follow a Gaussian distribution. In contrast, the RSRP values mainly fall in the range between

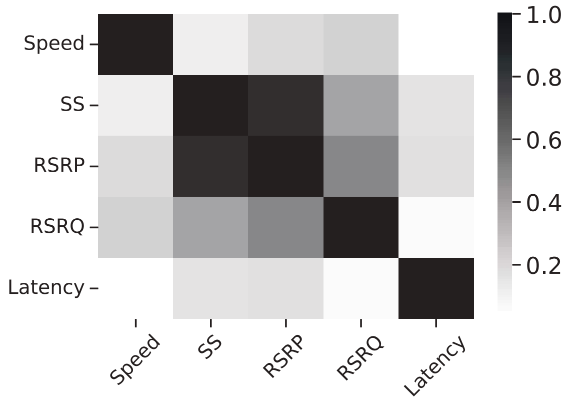


Fig. 6. Correlation between the latency and other metrics.

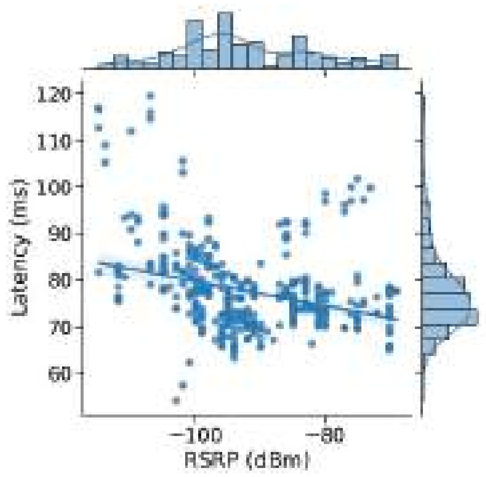


Fig. 7. Joint distribution between the latency and RSRP.

-110 and -70 dBm, with a larger variation than the latency. Based on these observations, we draw a linear-regression line according to the root mean square error minimization rule. We can observe that most points are distributed near the regression line, and the point with a lower RSRP has a higher expected latency. This indicates that signal strength is correlated with the latency and can be used to approximate the latency.

We consider three metrics that are related to signal strength. SS measures the received power of the LTE signal from the serving BS. RSRP is the average power of cell-specific reference signals over multiple LTE resource blocks within the measurement frequency band. Thus, RSRP represents the signal power over the bands of interest. RSRQ is ratio of the carrier power to the interference power. RSRQ is an important parameter for handover decision. However, the correlation between the latency and any single metric (SS, RSRP, or RSRQ) is only about 0.4, which is not sufficient to be employed as a feature for time series prediction. To take advantage of these strength metrics from different measurement perspectives, we propose to preprocess the data using SS, RSRP, and RSRQ.

B. Data Clustering for Latency Prediction

We apply the K-medoids [29] approach to classify the latency measurements according to their corresponding SS, RSRP, and RSRQ. K-medoids is a distance-based clustering technique that splits the input dataset $\mathcal{S}$ into K groups, as follows:

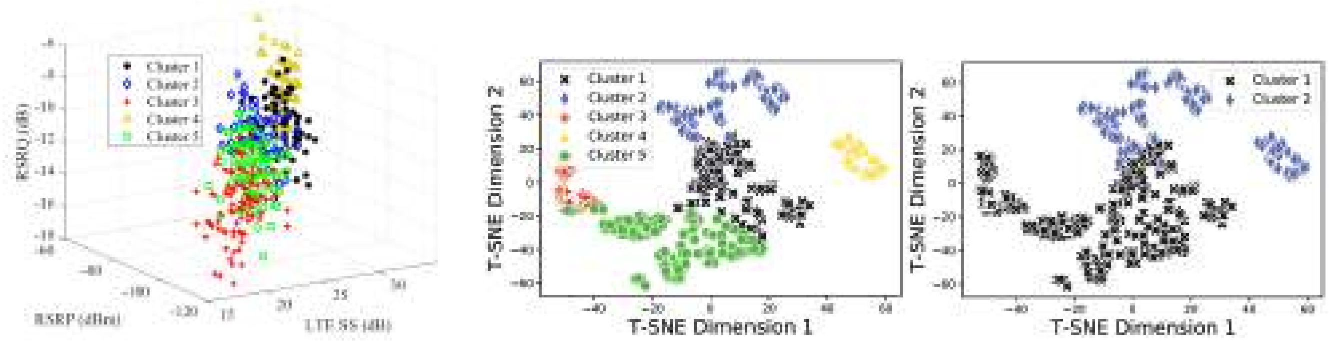
1) Randomly generate K medoids and compose the medoids set $\mathcal{M}$ in the three-dimensional dataset $\mathcal{S}$, according to SS, RSRP, and RSRQ of the input;
2) Calculate the Euclidean distances between all the data points in $\mathcal{S}$ and the medoids in $\mathcal{M}$. Then, cluster each data point to the medoid with the minimum distance;
3) In each cluster, test each data point as a potential medoid and calculate if the average distance is reduced;
4) If so, reassign data points in $\mathcal{S}$ to the updated $\mathcal{M}$;
5) Repeat steps 2 through 4 until the minimum average distance between each point and the corresponding medoid is achieved.

By employing the K-medoids algorithm, the input dataset $\mathcal{S}$ is clustered into K groups according to SS, RSRP, and RSRQ. In contrast to traditional supervised machine learning methods (e.g., support vector machine (SVM) and multi-layer perceptron (MLP) [30]), the K-medoids algorithm can split data without prior knowledge of their labels. Therefore, such an unsupervised learning algorithm is more suitable for preprocessing the data and finding the best clustering strategy in dynamic network environments. Fig. 8(a) depicts an example of applying the K-medoids clustering approach with K = 5. It can be seen that the three signal strength parameters are consistent, i.e., the signal becomes stronger when going from the lower left to the upper right. The clusters with a better signal quality are more likely to achieve lower latency (e.g., the cluster with yellow triangles has lower latency values than other clusters).

We also use the t-distributed stochastic neighbor embedding (t-SNE) [31] approach to project the feature and visualize the clusters in 2D space. T-SNE is a statistical method for visualizing high-dimensional data by giving each data point a location in a low-dimensional map. Therefore, T-SNE projection only changes the relationship between the point and cluster, and does not change the point distribution. By t-SNE with cluster number of five, we observe that data points with similar features form distinct clusters, as shown in Fig. 8(b). This indicates intrinsic differences in the collected data and justifies the use of clustering.

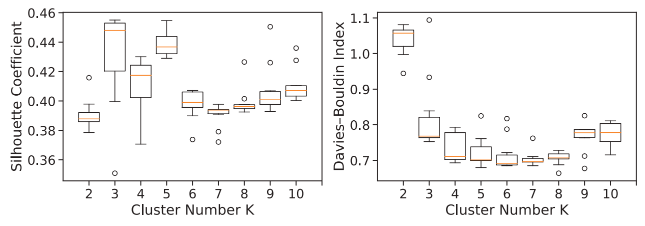
C. Impact of Clustering Number

We also illustrate the points distribution in the projected t-SNE dimensions in the case of two clusters. In this case, the yellow-colored points in Fig. 8(b) are merged into another cluster in Fig. 8(c), although the points in the merged cluster are not close to each other. This indicates that clustering measured data into two groups is insufficient. To quantify the impact of cluster number K, we introduce two coefficients to evaluate the impact of clustering: *Silhouette Clustering Coefficient* [32] and *Davies–Bouldin Index* [33]. The silhouette coefficient measures how similar an object is to its cluster (cohesion) compared to other clusters (separation). The silhouette coefficient ranges from -1 to +1, where a high value indicates that the object is well matched to its cluster and



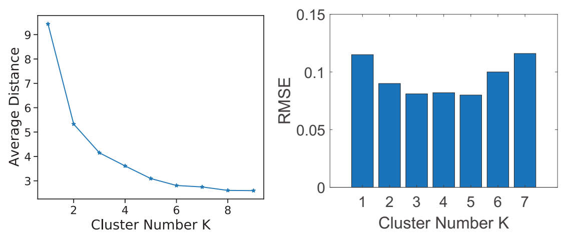
(a) Original features with five clusters (b) Projected features with five clusters (c) Projected features with two clusters

Fig. 8. Clustering latencies according to RSRP, RSRQ, and SS (K-medoids algorithm) and projection of signal strength metrics with t-SNE.



(a) Silhouette Clustering Coefficient (b) Davies–Bouldin Index

Fig. 9. The clustering coefficients vs. cluster number K.



(a) Average distance vs. cluster number (b) RMSE vs cluster number

Fig. 10. K-medoids distance and the prediction error of LSTM network under different cluster numbers.

poorly matched to neighboring clusters. The Davies–Bouldin index is denoted as the ratio of the within cluster scatter, to the between cluster separation. Thus, clusters farther apart and less dispersed will result in a lower score. The minimum score is zero, with lower values indicating better clustering.

We consider a range of cluster numbers from two to ten, and calculate the corresponding two coefficients. We repeat the simulation ten times and depict the boxplot that shows the $25\% - 75\%$ confidence interval and the mean value. In Fig. 9(a), the Silhouette coefficient has larger mean values when $K = 3$ and 5. In Fig. 9(b), the expected Davies–Bouldin index first decreases when $K < 6$ and increases with K afterward. Based on the observation in Fig. 9(a), $K = 5$ is the best choice for our case. In addition to the above observation, we also find that compared to other K values, the clustering coefficients have a more significant change when K increases from 2 to 3. This indicates intrinsic differences in the collected data and necessitates the clustering. When ranging K from 2 to 5, the Silhouette coefficient is increased from 0.391 to 0.438 while the Davies–Bouldin index is decreased from 1.042 to 0.725, showing that our clustering approach can effectively separate the data.

To further compare different settings of cluster number K, we use the average distance between any point and its corresponding center as the metric. Fig. 10(a) depicts the average distance under different cluster numbers. The distance decreases as K increases, with a slower rate when K is larger. This indicates that increasing K will not significantly reduce the average distance after K reaches a certain value. Note that the average distance is only one of many metrics that is related to the prediction performance. To show the prediction accuracy under varying values of K, we use the Root Mean Square Error (RMSE) to measure the errors between the actual and predicted value for LSTM networks. In Fig. 10(b), we show the RMSE of the LSTM predictor under different numbers of clusters (K). The prediction error is reduced when the K is between two and five but increases when $K > 5$, which is resulted from overfitting. Accordingly, we set $K = 5$ in our prediction model.

D. LSTM Network Design

Our measurements reveal that the latency at a given time is impacted by both short-term and long-term historical values, and it follows a trackable pattern over time once the small-scale variations are filtered out. This motivates us to employ an LSTM network, a kind of recurrent neural network (RNN) architecture, to estimate the latency. A initial attempt at designing such a model was presented in [1]. In here, we fine-tune this design and study its transient behavior.

The LSTM network stores the memory in the cell states and controls the information flow (i.e., determines what to remember and what to forget) by adapting the parameters of several *gating functions*, and can have a shallower structure compared to other deep neural networks, 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 LSTM cells. Let x_i be the ith measured latency, and n the total number of measurements. Let $g(\cdot)$ be the cluster function, which can be expressed as:

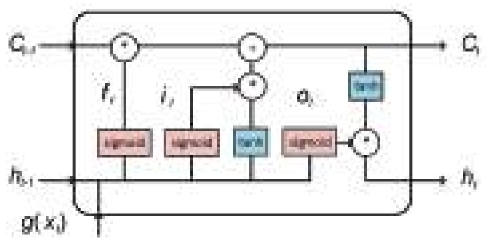


Fig. 11. Architecture of an LSTM cell.

$g(x_i) = x_{k,i}$, if x_i belongs to cluster k, $k = 1, 2, ..., K$. After clustering, inputs are divided into K groups: X_1, X_2,..., X_K. During the training phase, K different LSTM networks are trained using X_k, $k = 1, 2, ..., K$. During the testing phase, the clustering algorithm decides which cluster to use and then selects the corresponding LSTM network to predict the latency.

A typical LSTM network consists of multiple LSTM cells that are used to learn $f(\cdot)$ during the training phase. Fig. 11 shows the architecture of one LSTM cell. At each time step t, $g(x_t)$ is the classified input and h_t is the cell's output. The cell's output from the previous time step, h_{t-1}, is combined with the current input $g(x_t)$ and fed into the current cell. We further denote the cell state as C_t. This state records the cell parameter and updates at each step. Several gates, including an input gate (i_t), output gate (o_t), and forget gate (f_t), are applied to alleviate the gradient vanishing and exploding impacts on the RNN. These gates employ sigmoid (σ) as the activation function, which generates an output between zero and one. The output of an intermediate cell state at time t ($\widetilde{C}_t$) updates similarly with gates. The current inputs $g(x_t)$ and previous cell state C_{t-1} are processed by a hyperbolic tangent function that generates an output between -1 and 1:

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

where, W_c is the weight for the cell state, U_c is the recurrent weight, and b_c is the bias. The element-wise product of the updated $\widetilde{C}_t$ and i_t is regarded as the first part to update C_t. The other part comes from the multiplication of C_{t-1} and f_t. With this structure, the cell state can adjust the impact of the current input and the previous state with the input and forget gates, and generate the output of the cell h_t, which will be used for the next time step:

$$h_t = o_t * \tanh(i_t * \widetilde{C}_t + f_t * C_{t-1}). \tag{2}$$

E. LSTM Network Training and Testing

The training process involves hyperparameter tuning, which is performed offline. After training is completed, the prediction will be conducted online using the trained LSTM networks. Before feeding the data into the neural network, we normalize them as follows:

$$x_i{}' = \frac{x_i - \bar{x}}{\sum_{i=1}^{n} \frac{1}{n}(x_i - \bar{x})^2}, \text{ for } i = 1, 2, ..., n \tag{3}$$

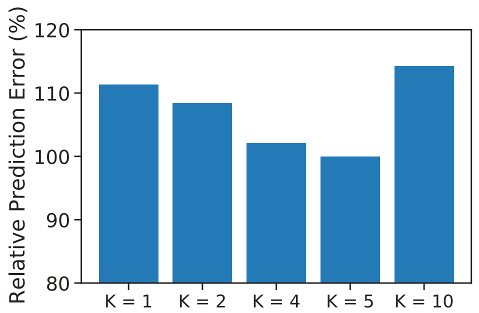


Fig. 12. Relative prediction error under different numbers of clusters using 100 LSTM cells.

where $\bar{x}$ is the sample mean latency of the input data. The normalized data are then split into 80% for training and 20% for testing. Our model is a sequential model and the latency prediction depends only on historical latency data. The initial learning rate is set to 0.005 and the learning rate drop factor is set to 0.2, so that we can limit the impact of the first several samples in case these samples do not represent the common feature.

We first set K independent LSTM networks, where each network is trained using data from one of the clusters produced by the K-medoids algorithm. The number of cells for each network is set to 20. Then, we train this model to find the optimal regression parameter and predict the latency based on input sequences. The difference between the prediction and the actual latency can be regarded as the loss function $\mathcal{L}$. Thus, in each training iteration, the weights and bias can be updated by the back-propagated gradients calculated from $\mathcal{L}$. We finally monitor the loss of RMSE and stop the training when $\mathcal{L}$ cannot be further reduced.

We study the prediction error under different numbers of cells and clusters for the given group in a controlled way, as shown in Fig. 12. In particular, we fix the number of LSTM cells at 100, which matches the total number of cells of the five-cluster predictor. We can observe that even though the error decreases as K increases from 1 to 4, this error is still higher than the one in the benchmark case. The error rebounds to a higher value when $K = 10$. Therefore, for the rest of the paper, we use $K = 5$ and 20 cells in each LSTM network.

F. Transient Behavior Analysis for LSTM Network

In the conventional machine learning algorithm, the training and the testing data usually share some common features. We train the neural network to learn these features from the training dataset and then use the trained model to evaluate the testing dataset. However, the communication environment could be various, especially in mobile edge computing systems, and such changes can degrade the prediction accuracy of the LSTM network. For the time series regression problem, a machine learning algorithm usually uses historical data as the input to predict the value for the next step(s). Since the data are aligned in chronological order, the first point in the testing dataset usually relies on the previous data. Nonetheless, such an assumption is only sometimes true in the mobile case. The point in the testing part could have a different feature from the previous measurements. As a result, we consider using zero instead of historical data at the beginning of the testing and changing the testing data by adding noise with different deviations to evaluate the transient behavior of the LSTM network in Section VIII.

VI. Prediction of the Residual Component

A. Predictor Design

The residual component that remains after subtracting the baseline is modeled using a hybrid statistical approach that combines three predictors.

1) PDF Approximation: The distributions of the RTT latency between the user and the edge/cloud servers, previously shown in Fig. 2, can be approximated by a kernel density. The latency shows a higher density when it is closer to the mean value and becomes more sparse when it is further away. Accordingly, we propose a prediction approach based on the sampled value of the kernal-based PDF. However, such a PDF evolves as new measurements are received; thus, the real distribution is hard to capture at a given time. We investigate and evaluate several methods for finding good density estimation of the measured latency. Specifically, the probability that the latency equals to x is calculated using the following Epanechnikov Kernel function:

$$\widehat{f}_D(x) = \frac{1}{N}\sum_{i=1}^{N}\mathcal{K}_D\left(x - x_i\right) = \frac{1}{ND}\sum_{i=1}^{N}\mathcal{K}\left(\frac{x - x_i}{D}\right) \quad (4)$$

where x_i is the ith data point, N is the number of samples used to generate the PDF approximation, and $\mathcal{K}_D$ is a scaled Epanechnikov Kernel function with a smoothing parameter D. By using the Kernel function, we obtain the PDF of x as a continuous variable.

From (4), we see that $\widehat{f}_D(x)$ depends on N and the selection of the sample set (i.e., x_1, x_2,..., x_N). Data used to compute $\widehat{f}_D(x)$ is obtained by sliding a window over the measurement trace. Because more recent samples contain more valuable information, we include two predictors that take advantage of both short- and long-term information: y_{STS} and y_{LTS}. These predictors are defined based on the given sampling range.

2) Moving Average: The above sampling-based approach cannot fully capture dependencies in the observed latencies (i.e., the autocorrelations between samples). Therefore, we augment them with an exponentially weighted moving average (EWMA) predictor:

$$y_i = y_{i-1} + \alpha(x_i - y_{i-1}) \quad (5)$$

where y_i is the prediction at time i and α is the weight parameter. This α can be determined by $\alpha = \frac{2}{N+1}$. We denote the EWMA predictor by y_{MA}.

B. Combining Predictors

From the measurements, we observed that the accuracy of different predictors varies with the measuring periods. For example, y_{STS} can have better performance than the other two when the latency fluctuates significantly. This is because recent latencies have more impact on the current latency than y_{LTS}, and the fluctuation further drops the accuracy of the EWMA approach. Hence, we propose to combine these three predictors through dynamic weights a, b, and c, i.e., the weights (coefficients) can be adapted based on their relative importance and reduction in the prediction error by one predictor. Note that y_{STS}, y_{LTS}, and y_{MA} are relatively independent in the prediction. As a result, the final prediction value can be denoted as:

$$y_i = a\, y_{\mathrm{STS}} + b\, y_{\mathrm{LTS}} + c\, y_{\mathrm{MA}}. \quad (6)$$

Where $a + b + c = 1$.

C. Adaptation of Coefficients

To improve the prediction accuracy under various dynamic environments, we update the coefficients a, b, and c with time. Specifically, at each time step i, we first set a potential coefficient set including J groups of coefficient vectors and calculate the possible predictions $y_{i,1}, y_{i,2}, ...y_{i,j}, ..., y_{i,J}$ as follows:

$$\begin{bmatrix} y_{\mathrm{STS}}, & y_{\mathrm{LTS}}, & y_{\mathrm{MA}} \end{bmatrix} \cdot \begin{bmatrix} a_1 & a_2 & ... & a_j & ... & a_J \\ b_1 & b_2 & ... & b_j & ... & b_J \\ c_1 & c_2 & ... & c_j & ... & c_J \end{bmatrix}. \quad (7)$$

Where (a_j, b_j, and c_j) is the possible coefficient vector (a, b, and c) at time i. Then, we calculate the error vector $\begin{bmatrix} E_1, & E_2, & ......, & E_J \end{bmatrix}$ between $y_{i,j}$ and real latency x_i, and find $j' \in \{1, 2, ..., J\}$ that minimizes E_j:

$$j' = \arg \min_{j' \in \{1,2,3,...,J\}} E_j \quad (8)$$

Finally, we update $\{a_{i+1}, b_{i+1}, c_{i+1}\} = \{a_{j'}, b_{j'}, c_{j'}\}$ for the next step prediction.

VII. Latency Estimation-based Task Offloading

In a V2X system, a user usually keeps sending packets before receiving results from the server. For example, an autonomous vehicle may collect data from different sensors (e.g., camera, lidar, or radar) in real-time while these data are being processed. The vehicle continues to offload tasks to the server. As a result, tasks offloaded to the same server can create a queue, whose delay contributes to the E2E delay. The previous section aims at predicting the RTT, which only includes access and propagation delays. To model the total packet delay in the V2X system, other latency components, including transmission delay, queuing delay, and processing delay, should also be incorporated. Fig. 13 shows the offloading process. The propagation and transmission time accounts for the packet delivery time, while the queueing and processing time depends on the type of offloaded tasks and the server.

A. Latency Model

In our model, the E2E delay consists of multiple delay components. Since the typical size of a ping packet is only 56 bytes, we ignore the transmission and processing times of the ping packet, and use the measured RTT to represent the

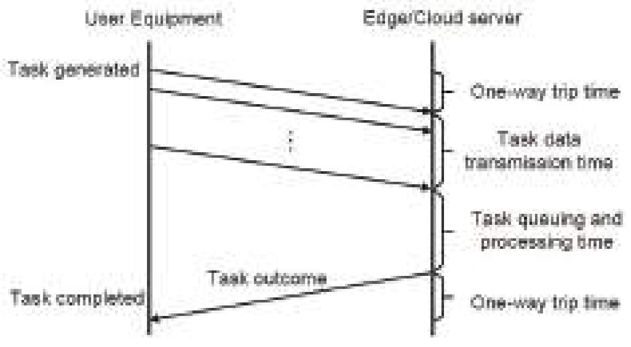


Fig. 13. Latency components of the communication between UE and the server.

propagation delay (τ_{prop}). A packet offloaded to the remote server traverses more hops than a packet sent to an edge server, so the average transmission time to the CN is expected to be larger than the EN. Formally, we define the transmission delay (τ_{tran}) as the total time required to transmit the data associated with a task to the designated server.

Let $\tau_{tr}^{(C)}$ and $\tau_{tr}^{(E)}$ be the average transmission delay for cloud and edge nodes, respectively. Define the node selection parameter:

$$\delta_i \triangleq \begin{cases} 1, & \text{if the } i\text{th packet is offloaded to the edge node} \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Accordingly, the transmission delay for the ith packet is $\tau_{tr,i}^{(C)}$ if offloaded to the CN and is $\tau_{tr,i}^{(E)}$ if offloaded to the EN. Therefore, the overall transmission delay for the ith packet is given by:

$$\tau_{\text{tran},i} = (1-\delta_i)\tau_{tr,i}^{(C)} + \delta_i \tau_{tr,i}^{(E)}. \tag{10}$$

These transmission delays can be calculated with the timestamps already contained in the packet's header by CN/EN [34]. In the simulation, we assume that the transmission time ($\tau_{tr,i}^{(C)}$ and $\tau_{tr,i}^{(E)}$) varies but follows an exponential distribution with effective link rate parameters $R_{tr}^{(C)}$ and $R_{tr}^{(E)}$, where $\frac{1}{R_{tr}^{(C)}}$ and $\frac{1}{R_{tr}^{(E)}}$ are the mean transmission time for the unit size packet.

After a packet arrives the server, the queuing delay (τ_{queue}) is the time that the packet/task waits in a queue at a CN/EN before its processing begins. We build two virtual queues for the EN and CN to simulate the packets processing at the server. Packets will be queued to the EN or CN by offloading decisions. In our problem setting, variations in the queuing delay are caused by the randomness of the offloading process between the CN and EN and the variation of service time for different tasks. We assume that the CN has a sufficiently large buffer size such that there is no overflow, whereas the EN has a relatively small buffer size. We further assume first-come-first-serve (FCFS) service discipline. The interarrival time between two tasks is set to 500 ms, which is the same as the one used in the latency measurements. The service times are modeled as i.i.d. exponentially distributed random variables with rate ν (ν is the inverse of the mean service time). Note that the packet arriving rate interacts with the estimated queueing and

Algorithm 1 Latency estimation-based offloading algorithm

```
Input: $\vec{M}$, $R_{tr}^{(E)}$, $R_{tr}^{(C)}$, $P$, $R_{pr}^{(E)}$, $R_{pr}^{(C)}$
Output: $\vec{\delta}$
    initialization:
    Round $r$ = 1, Time $i$ = 0, $\vec{\tau}_{\text{prop},r}^{(E)}, \vec{\tau}_{\text{prop},r}^{(C)} \leftarrow$ pred ($\vec{M}$),
    $\vec{\tau}_{\text{tran},r}^{(E)}, \vec{\tau}_{\text{tran},r}^{(C)}, \vec{\tau}_{\text{queue},r}^{(E)}, \vec{\tau}_{\text{queue},r}^{(C)}, \vec{\tau}_{\text{proc},r}^{(E)}, \vec{\tau}_{\text{proc},r}^{(C)} \leftarrow \vec{0}$,
    $\vec{\delta}_{r-1} \leftarrow -\vec{1}, \vec{\delta}_r \leftarrow \vec{0}$, R = Max Iteration
 1: for $r = 1$ to R do
 2:   if $\mathcal{H}(\vec{\delta}_{r-1}, \vec{\delta}_r) \geq \alpha L$ then
 3:     $\vec{\delta}_r \leftarrow \vec{\delta}_{r-1}$
 4:     for $i = 0$ to $L-1$ do
 5:       if $\tau_{\text{totl},r,i}^{(E)} \leq \tau_{\text{totl},r,i}^{(C)}$ then
 6:         $\delta_{r,i} = 1$
 7:       else
 8:         $\delta_{r,i} = 0$
 9:       end if
10:       $\tau_{\text{tran},r,i+1}^{(E)}, \tau_{\text{queue},r,i+1}^{(E)}, \tau_{\text{proc},r,i+1}^{(E)} \leftarrow$ sim (EN, $\vec{\delta}_r$)
11:       $\tau_{\text{tran},r,i+1}^{(C)}, \tau_{\text{queue},r,i+1}^{(C)}, \tau_{\text{proc},r,i+1}^{(C)} \leftarrow$ sim (CN, $\vec{\delta}_r$)
12:       $\tau_{\text{totl},r,i+1}^{(E)} = \tau_{\text{prop},r,i+1}^{(E)} + \tau_{\text{trans},r,i+1}^{(E)} + \tau_{\text{queue},r,i+1}^{(E)} + \tau_{\text{proc},r,i+1}^{(E)}$
13:       $\tau_{\text{totl},r,i+1}^{(C)} = \tau_{\text{prop},r,i+1}^{(C)} + \tau_{\text{trans},r,i+1}^{(C)} + \tau_{\text{queue},r,i+1}^{(C)} + \tau_{\text{proc},r,i+1}^{(C)}$
14:     end for
15:   else
16:     return $\vec{\delta}_r$
17:   end if
18: end for
19: return $\vec{\delta}_r$
```

processing time in our queueing system, so the queue is not a $M/M/1$ model. Thus, we can not rely on the numerical results from the existing queueing models. Instead, we need to dynamically simulate the packets arriving associated with the offloading schemes to model the waiting time.

The actual service time is represented by the processing delay (τ_{proc}), which can be calculated from the packet size P_i, the task computational complexity z_i, and the processing rate R_{pr}, as: $\tau_{\text{proc}} = \frac{P_i z_i}{R_{pr}}$. Thus, for the ith packet, the processing delay can be written by:

$$\tau_{\text{proc},i} = \frac{P_i z_i}{(1-\delta_i)R_{pr,i}^{(C)} + \delta_i R_{pr,i}^{(E)}} \tag{11}$$

where z_i represents the required CPU cycles for processing one bit of data. $R_{pr,i}^{(C)}$ and $R_{pr,i}^{(E)}$ are the actual computational capabilities for P_i at CN and EN, respectively, which are measured in CPU cycles per second. Similar to the transmission time setting, the processing time ($\frac{1}{R_{pr,i}^{(C)}}$ and $\frac{1}{R_{pr,i}^{(E)}}$) is modeled as i.i.d. exponentially distributed random variables with rate $R_{pr}^{(C)}$ and $R_{pr}^{(E)}$.

Combining all delay components, the total delay for the ith packet can be determined: $\tau_{\text{totl},i} = \tau_{\text{prop},i} + \tau_{\text{tran},i} + \tau_{\text{queue},i} + \tau_{\text{proc},i}$, where $\tau_{\text{prop},i}$, $\tau_{\text{tran},i}$, and $\tau_{\text{proc},i}$ are determined by the selected server, while $\tau_{\text{queue},i}$ is determined by the number of packets in the queue (which is impacted by the offloading choices of all connected devices). All the timestamps and rate parameters are available either in the received packet or on the server side. The simulation results for each received packet will be fed back to the user when the server sends the acknowledgment. It is evident that node selection plays an important role in determining network latency. Therefore, we design an algorithm for the node selection based on the approximated E2E delay, aiming to reduce overall latency.

B. Task Offloading based on Latency Prediction

To estimate the queueing delay of a sequence of packets, we extend the proposed prediction approach and predict the next L time slots propagation delay $\vec{\tau}_{\mathrm{prop}}$. After that, we model two queues at the CN and EN, and simulate the delays under different offloading schemes. We use the vector $\vec{\tau}_{\mathrm{prop}}$ to present $\{\tau_{\mathrm{prop},1}, \tau_{\mathrm{prop},2},..., \tau_{\mathrm{prop},L}\}$ which includes the delay at each time slot within L. Based on the predicted $\vec{\tau}_{\mathrm{prop}}$, we simulate $\vec{\tau}_{\mathrm{trans}}$, $\vec{\tau}_{\mathrm{queue}}$, and $\vec{\tau}_{\mathrm{proc}}$ and propose the computation offloading algorithm that aim to reduce vehicle user's E2E delay.

The propagation delay vector for CN $\vec{\tau}^{(C)}_{\mathrm{prop}}$ and for EN $\vec{\tau}^{(E)}_{\mathrm{prop}}$ are predicted based only on historical measurements $\vec{M}$. Therefore, $\vec{\tau}^{(C)}_{\mathrm{prop}}$ and $\vec{\tau}^{(E)}_{\mathrm{prop}}$ will not change with the simulation iteration r (i.e., the propagation delay vectors for EN and CN at round r will be the same with round $r+1$: $\vec{\tau}^{(E),(C)}_{\mathrm{prop,r}}=\vec{\tau}^{(E),(C)}_{\mathrm{prop,r+1}}$). In the first round for node selection, we only consider $\vec{\tau}^{(E),(C)}_{\mathrm{prop,r}}$ for there is no prior knowledge about the queueing tasks in EN/CN. Thus, we always choose the node with the lower τ_{prop} between $\tau^{(E)}_{\mathrm{prop,r,i}}$ and $\tau^{(C)}_{\mathrm{prop,r,i}}$ and offload packets to this node for $i = 1, 2, ..., L$. Based on the node selection vector at round r ($\vec{\delta}_r$), we create two queues, one for CN and the other for EN. Then, $\vec{\tau}^{(E),(C)}_{\mathrm{trans,r}}$, $\vec{\tau}^{(E),(C)}_{\mathrm{queue,r}}$, and $\vec{\tau}^{(E),(C)}_{\mathrm{proc,r}}$ for these queues at round r can be simulated. Similar with round 1, we let each packet choose a node with lower τ_{totl} between $\tau^{(E)}_{\mathrm{totl,r,i}}$ and $\tau^{(C)}_{\mathrm{totl,r,i}}$ and find the latency-optimal selection vector $\vec{\delta}_r$ for these L time slots at round r.

The simulation stops when there is no change in $\vec{\delta}_r$ or when a certain number of iterations is reached. Note that $\tau_{\mathrm{proc,r,i}}$ is a exponentially distributed random variable and may lead to a variation in $\delta_{r,i}$ during the iteration. This can result in the permutation of elements in $\vec{\delta}_r$ and make the algorithm hard to converge. In this case, we relax the stopping condition as the difference between $\vec{\delta}_r$ and $\vec{\delta}_{r+1}$ less than the tolerance: $\mathcal{H}(\vec{\delta}_r, \vec{\delta}_{r+1}) < \alpha L$. Where $\mathcal{H}$ is the Hamming distance as defined in [35], α is the tolerance rate, and L is the length of the selection vector. In our simulation, we set α and L as 0.05 and 100, respectively. The threshold is set to 500.

We assume that the server monitors the number of packets in the queues and feed back such information to the UE every time receiving them. For example, suppose that in round $r-1$, the 10th packet is to be offloaded to the CN, i.e., $\delta_{r-1,10} = 0$. P_{10} also knows the queuing decisions of the first 9 packets. In such a situation, if there are 3 packets queueing at the CN but only 1 packet queueing at the EN, UEs will have a high probability to choose EN as the offloading node to reduce the latency. Then, we need simulate the E2E delay for the cloud node $\tau^{(C)}_{\mathrm{totl,r,10}}$ and for the edge node $\tau^{(E)}_{\mathrm{totl,r,10}}$ based on the current queues. If $\tau^{(E)}_{\mathrm{totl,r,10}} < \tau^{(C)}_{\mathrm{totl,r,10}}$, the selection variable $\delta_{r,10}$ will be updated to 1. Note that the update of $\delta_{r,10}$ will influence the queues in the future moments. Therefore, we need to evaluate the impact on the whole sequence of selection vector, i.e., from $\delta_{r,1}$ to $\delta_{r,L}$. After that, the node selection vector will be updated from $\vec{\delta}_r$ to $\vec{\delta}_{r+1}$. The latency prediction-based offloading algorithm is summarized in Algorithm 1.

TABLE II
PARAMETERS SETTING

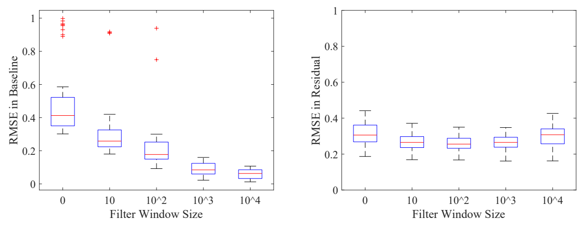
Parameters	Settings
Transmission interval, τ_{int}	500 ms
Input data size, P_i	[500 – 3000] KB
Computational complexity, z_i	2000 CPU cycle/bit
EN transmission rate, $R^{(E)}_{tr}$	5 Mbps
CN transmission rate, $R^{(C)}_{tr}$	50 Mbps
EN computation rate, $R^{(E)}_{pr}$	2×10^{10} CPU cycle/s
CN computation rate, $R^{(C)}_{pr}$	8×10^{10} CPU cycle/s

VIII. Performance Evaluation

In this section, we evaluate the performance of the proposed latency prediction approach and the corresponding offloading scheme. The access delay is measured by *Delay Explorer* as described in Section III. We first predict the propagation delay. The mean and STD for the prediction error are compared with other estimation techniques. To show the impact of modeling the propagation delay using two components, the accuracy of different window sizes is studied. After predicting the propagation delay, we simulate the offloading procedure between UE, CN, and EN, where we set the CN's a computational capacity to 8×10^{10} CPU cycles/s and the EN's computational capacity to 2×10^{10} CPU cycles/s. The data size per task ranges from 500 KB to 3000 KB. Other simulation parameters are referred to [10], [11] and given in Table II. Under this setting, we simulate the transmission delay, queuing delay, and processing delay. We then calculate the corresponding offloading profile, as in Algorithm 1. The few works in the literature that considered tasks' queueing delay (e.g., [9]) did not incorporate E2E latency estimation in task assignment. Hence, to evaluate our proposed scheme, we compare it with CN-only, EN-only, and random selection schemes, considering the average E2E delay and the probability of the successful transmission.

A. Impact of Filter Settings

For we extract the trends component from our RTT measurements by a window-based average filter, we first evaluate the impact of such window size. Fig. 14 shows the RMSE for the predicted RTT under different window sizes. For the baseline part, the RMSE decreases quickly with the window size and then stabilizes. This is because the filter removes the rapidly varying component from the original data and makes the trends hidden under the data more trackable. In addition to the mean RMSE drop, the variation of RMSE also reduces with window



(a) Baseline part (b) Residual part

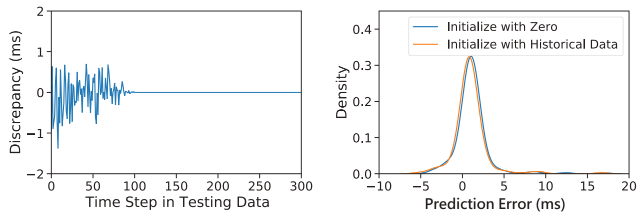
Fig. 14. RMSE for predicted propagation delay under different window sizes.

size, which indicates that our prediction performs more stable. Fig. 14(b) presents the impact of window size for the residual component. The RMSE decreases and then rebounds after the window size exceeds 1000. This is because the average over large window size cannot remove the short-term fluctuations of the data. By separating the noisy component from the trend, we remove the irrelevance and simplify the regression problem for the LSTM network. The refined data is more trackable and requires fewer computational resources. As a result, we design the lightweight LSTM network with high accuracy but a shallow structure.

B. Study of Transient Behavior on the LSTM Network

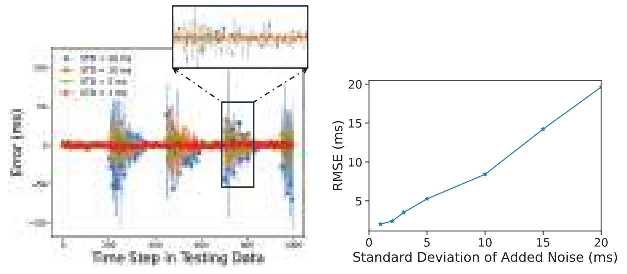
We study the transient behavior of the LSTM network while latency data are being measured during driving. Recall that we apply the LSTM predictor using an input window size of 100, and train it in a sequence-to-sequence fashion. We use the last value of the output sequence to make a latency prediction for the next step. Thus, the transient behavior occurs at the beginning of the prediction process due to the lack of historical data. In our previous testing results, the missing historical data were replaced by training data, until the first 100 latency values have been captured. To study the transient behavior due to this initialization process, we have conducted new experiments where we initialize the unknown data points during testing with zeros (i.e., we assume the predictor does not initially have any historical data). In this case, the first 100 predictions may be inaccurate. However, such impact is limited to the initialization phase. As new measurements become available to the predictors, they can be used for subsequent predictions. We visualize the differences between the two initialization schemes (i.e., are initialized with actual data and the other with zeros) in Fig. 15(a). We can observe that the impact of the transient behavior becomes quite negligible as time goes by. We also plot the distribution of the prediction errors for both initialization, in Fig. 15(b). We can see that the error distributions are very similar. In particular, the average difference in error is only 0.127 ms.

In addition, we study the impact of sudden changes in the observed latencies on the predictor's performance. To inject sudden changes in the data, we periodically add a Gaussian random noise into the testing data. We use 1000 data points during the testing phase as an example. The 1000 data points are divided into four sets of 250 points. For each set, we set 200 data points as ordinary data and we add Gaussian noise



(a) Discrepancy between predictions when the LSTM predictor is initialized with zeros or with training data (b) Distribution of the RMSE for the two initialization schemes

Fig. 15. Impact of the transient state during the testing of time-series latency prediction.



(a) Prediction error vs. time for different AWGN STDs (b) RMSE comparison under different STDs

Fig. 16. Impact of sudden changes in the network environment during testing time-series latency prediction.

to the remaining 50 data points to emulate sudden changes. We repeat this process for all four sets, so the testing data end up with four sudden changes. We fix the mean of the added Gaussian noise to zero to rule out bias, and then vary the STD to simulate different dynamics of change. The prediction error is defined as the difference between the predicted and real (measured) latency, and is visualized in Fig. 16(a). Expectedly, the prediction error increases when there is a sudden change in the data. This trend extends beyond 50 data points, to about 100 additional points. The reason is that the LSTM predictor will use the 50 noisy points as part of the input for predicting the next 100 latency values. We also observe that the prediction error is larger as the STD of noise increases. We summarize the RMSE under various STDs in Fig. 16(b). The RMSE approximately has a near-linear relationship with the STD of the noise.

C. Evaluation of the Proposed Prediction Approach

1) Metrics: To test the accuracy of our prediction approach, we set aside a fraction of the measurements. Measured latencies are collected at a fixed location and during drivings, as described previously. Both the mean prediction error ($\overline{e}$) and error STD (θ) are reported. We compare all the methods with a sample-mean predictor. For convenience, we define the ratio between $\overline{e}$ of the proposed method and the sample-mean predictor as ϵ, and denote the ratio between θ of the proposed method and the sample-mean predictor as μ.

2) Benchmarks: We developed a Kalman filter-based and particle filter-based models [36] to predict the latency. The Kalman Filter (KF) is the optimal Minimum Mean Square Error (MMSE) state estimator for linear Gaussian stochastic systems [37], [38]. We denote $\mathbf{r}$ is the measured latency samples and $\mathbf{x}$ is the predicted latency model, so we estimate $\mathbf{x_k}$ in time step k using the measured latency history from time step 1 to $k-1$. We first derive the distribution $p\left(\mathbf{x}_k\left|\mathbf{r}_{1:k}\right.\right)$ using the following standard Bayes rule

$$p\left(\mathbf{x}_k\left|\mathbf{r}_{1:k}\right.\right)=\frac{p\left(\mathbf{r}_k\left|\mathbf{x}_k\right.\right)p\left(\mathbf{x}_k\left|\mathbf{r}_{1:k-1}\right.\right)}{p\left(\mathbf{r}_k\left|\mathbf{r}_{1:k-1}\right.\right)} \tag{12}$$

where $p\left(\mathbf{x}_k\left|\mathbf{r}_{1:k-1}\right.\right)$ is given by

$$p\left(\mathbf{x}_k\left|\mathbf{r}_{1:k-1}\right.\right)=\sum p\left(\mathbf{x}_k\left|\mathbf{x}_{k-1}\right.\right)p\left(\mathbf{x}_{k-1}\left|\mathbf{r}_{1:k-1}\right.\right), \tag{13}$$

and $p\left(\mathbf{r}_k\left|\mathbf{r}_{1:k-1}\right.\right)$ and $p\left(\mathbf{r}_k\left|\mathbf{x}_k\right.\right)p\left(\mathbf{x}_k\left|\mathbf{r}_{1:k-1}\right.\right)$ is calculated from the histogram of our latency measurements.

The KF algorithm can now be rewritten as the following recursive

$$\begin{aligned}
p\left(\mathbf{x}_{k-1}\left|\mathbf{r}_{1:k-1}\right.\right) &= \mathcal{N}\left(\mathbf{x}_{k-1};\mathbf{m}_{k-1|k-1},\mathbf{P}_{k-1|k-1}\right), && (14)\\
p\left(\mathbf{x}_k\left|\mathbf{r}_{1:k-1}\right.\right) &= \mathcal{N}\left(\mathbf{x}_k;\mathbf{m}_{k|k-1},\mathbf{P}_{k|k-1}\right), && (15)\\
p\left(\mathbf{x}_k\left|\mathbf{r}_{1:k}\right.\right) &= \mathcal{N}\left(\mathbf{x}_k;\mathbf{m}_{k|k},\mathbf{P}_{k|k}\right), && (16)
\end{aligned}$$

where $\mathcal{N}\left(\mathbf{x};\mathbf{m},\mathbf{P}\right)$ is the Gaussian distribution with argument $\mathbf{x}$, mean $\mathbf{m}$ and covariance $\mathbf{P}$ defined as

$$\mathbf{P}_k=\mathbb{E}\left(\left(\mathbf{x}_k-\mathbf{m}_k\right)\left(\mathbf{x}_k-\mathbf{m}_k\right)^T\right). \tag{17}$$

Our preliminary model is a a linear system given by

$$\mathbf{r}_k=\mathbf{H}\mathbf{x}_k+\mathbf{w}_k, \tag{18}$$

where $\mathbf{H}$ is the observation transition matrix. $\mathbf{w}$ is the measurement noise assumed to follow a Gaussian distribution with zero-mean and covariance matrix $\mathbf{R}$.

The prediction step of KF is shown as follows

$$\begin{aligned}
\mathbf{x}_k^- &= \mathbf{F}\mathbf{x}_{k-1}+\mathbf{v}_{k-1}, && (19)\\
\mathbf{P}_k^- &= \mathbf{F}\mathbf{P}_k\mathbf{F}^T+\mathbf{Q}, && (20)
\end{aligned}$$

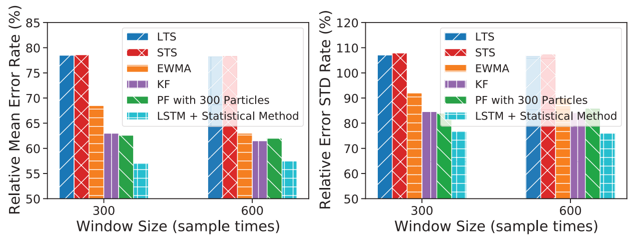
where $\mathbf{F}$ is the state-transition matrix, $\mathbf{v}$ is the noise and assumed to be zero-mean Gaussian, and $\mathbf{Q}$ is the covariance matrix of the process noise.

The update step of KF is shown as follows

$$\begin{aligned}
\mathbf{K}_k &= \mathbf{P}_k^-\mathbf{H}^T\left(\mathbf{H}\mathbf{P}_k^-\mathbf{H}^T+\mathbf{R}\right)^{-1}, && (21)\\
\mathbf{x}_k &= \mathbf{x}_k^-+\mathbf{K}_k\left(\mathbf{r}_k-\mathbf{H}\mathbf{x}_k^-\right), && (22)\\
\mathbf{P}_k &= \left(\mathbf{I}-\mathbf{K}_k\mathbf{H}\right)\mathbf{P}_k^-, && (23)
\end{aligned}$$

where $\mathbf{P}_k$ is the state covariance matrix. $\mathbf{K}_k$ is the Kalman Gain matrix which corrects the predicted prior state pdf for deriving the posterior. $\mathbf{x}_k^-$ and $\mathbf{P}_k^-$ are the prior state and covariance.

One shortcoming of KF is that it assumes the noise of measurement and observation to be Gaussian, which does not necessarily apply to our collected data. To overcome this limit, we apply the Particle Filter (PF) [39]–[41], to estimate the posterior density of state variables with given observation variables. In contrast to the KF in Equations (12)–(16), let $\{\mathbf{x}_{0:k}^i, w_k^i\}_{i=1}^{N_s}$ denote a random measure that characterizes the posterior pdf $p(\mathbf{x}_{0:k}|\mathbf{r}_{1:k})$, where $\{\mathbf{x}_{0:k}^i, i=1,...,N_s\}$ is a set of support particles with associated weights $\{w_k^i, 1,...,N_p\}$ and $\mathbf{x}_{0:k}=\{\mathbf{x}_j, j=0,...,k\}$ is the set of all states up to time



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

Fig. 17. Prediction error vs. window size under different prediction methods using measured traces.

k. The weights are normalized such that $\sum_i w_k^i=1$. Then, the posterior density at can be approximated as:

$$p(\mathbf{x}_{0:k}|\mathbf{r}_{1:k})\approx\sum_{1}^{N_s}w_k^i\delta(\mathbf{x}_{0:k}-\mathbf{x}_{0:k}^i). \tag{24}$$

Where $\delta(\cdot)$ is the Dirac delta measure. We end up with a discrete weighted approximation to the true posterior, $p(\mathbf{x}_{0:k}|\mathbf{r}_{1:k})$. More details about PF approach can be found in references [39]–[41].

3) Results Comparison: Relative prediction error $\overline{e}$ and STD μ of different methods are compared in Fig. 17. The structured LTS and STS methods have similar performance and can reduce ϵ by around 20%, but both methods raise μ about 8%. It indicates weak performance if we depend only on sampling predictors. Conventional approaches such as Kalman Filter (KF) and Particle Filter (PF) can achieve lower ϵ and μ than the sampling methods. However, μ for both predictors is still high and can reach around 85%, indicating that their prediction may not be steady in all scenarios. Our proposed approach can reduce ϵ by 45% compared to the sample-mean predictor. In addition, μ of the proposed approach is approximately 10% less than KF and PF, showing that our approach can achieve the best performance in terms of both precision and stability among all structured methods.

D. Analysis for Components in E2E Delay

We plot in Fig. 18 the fraction of each type of latency, relative to the total delay. The propagation delay represents about 40% of the total delay when the packet size is 500 KB. As the packet size increases, the propagation delay becomes less significant. When the packet size reaches 3000 KB, the propagation only takes about 10% of the total delay. In contrast, the queueing delay becomes more significant with the increase in the packet size. This is because the packet with a larger size requires more time to be processed under a given processing capacity, which forces latter packets to be queued before being served. The queueing delay's portion reaches almost 80% of the total delay when the packet size is around 3000 KB. The processing and transmission delays represent about 20% together, which is steady over different packet sizes. The results show that each component of the E2E delay takes a considerable percentage and is not neglectable.

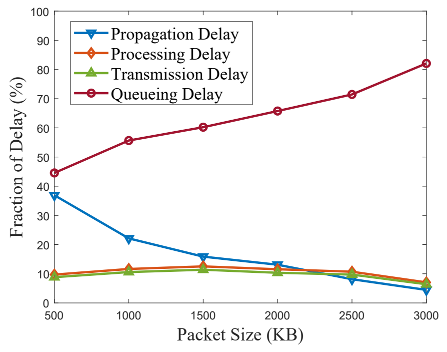


Fig. 18. The percentage of each type of latency in the total delay.

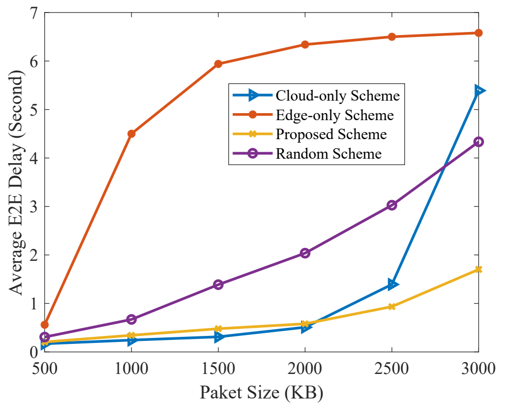


Fig. 19. Average E2E delay vs. packet size under different offloading schemes.

E. Evaluation of the Proposed Offloading Scheme

After predicting τ_{prop}, we apply the proposed algorithm to select between the CN and EN and reduce E2E delay, as described in Section VII. The CN is assumed to have infinite buffer size, while the EN has a maximum buffer size (B_{max}) of 7500 KB. The results are summarized in Fig. 19. As can be observed, for small data packets, the average E2E delay increases fast with the packet size [2] for the edge-only scheme. This is because the edge server has limited computing capacity, which makes its delay performance more sensitive to the packet size compared with the cloud server. As the packet size increases beyond 1500 KB, the E2E delay slows down. This indicates that the packets in the queue is getting close to B_{max}. In other words, most incoming tasks need to wait B_{max} packets to be processed in the queue. Given that the received packets exceed B_{max} will be discarded, the maximum queueing time will be achieved when packet size is large enough. In our case, queueing time approaches 6.5 seconds when packet size is 3000 KB. In contrast to the EN, the CN has a larger computing capacity. The E2E latency of the cloud-only scheme has a very low E2E delay when the packet size is less than 2000 KB. After that, the E2E latency starts to increase rapidly, which implies the computing capability cannot support the higher incoming rate, and the queueing delay plays a more important role than the processing delay. The random selection scheme selects the CN/EN servers randomly. It outperforms the cloud-only scheme only when the packet size exceeds the cloud server's computing capacity. In contrast to these schemes, our proposed approach can decide the offloading server based on the predicted latency. When the packet size is small, it has a similar E2E delay as the cloud-only scheme. When the packet size is large, our scheme can make use of the EN when CN is busy. This helps reduce the burden on the CN when there are packets queueing for service at the cloud server. The E2E latency of the proposed scheme is also more robust to variations in the packet size.

Fig. 20 shows the probability of success for different schemes. We say a transmission is successful when the data packet of a task is not discarded by the server. The CN has an infinite queueing capacity, so there no packets will be discarded. Therefore, the cloud-only scheme achieves 100% success probability no matter how large the packet size is. Although the CN does not discard the packet, its average E2E delay suffers significantly when packet size grows. This is because the later arrived packets have to wait for the server to process all the prior packets in the queue, and τ_{queue} is almost linearly increasing with the queued packet size. We indicate this trend in Fig. 19. As the packet size grows from 500 to 2000 KB, the EN's computing capability cannot serve requests on time, so data packets will be queued. The increase in the queue length results in reduced probability of success. The random scheme is greatly affected by the EN, for it still has 50% likelihood to choose the edge server even though the server has a low success rate. Contrary to a random choice, the proposed scheme can choose the server that is expected to have a shorter waiting time. Our scheme reduces the average E2E delay especially when packet size is large, without sacrificing too much on the probability of success. For example, when the packet size is 3000 KB, the proposed scheme has an average E2E delay of 1.5 seconds but the cloud-only one has an average delay of 5.5 seconds. The proposed scheme reduce the E2E delay by 72.7%. However, it only reduces about 3% probability of success compared to the cloud-only one. It means our scheme can efficiently make use of the CN to relieve the EN's computational burden. The results in Fig. 19 and Fig. 20 show that the proposed approach can maintain a lower average E2E delay and a higher probability of success at the same time. In our simulation, we found out that the proposed scheme results in an average E2E delay of about 38% of the random scheme when the packet size is between 1000 and 3000 KB.

IX. Conclusions

In this paper, we proposed a latency estimation-based offloading scheme for edge applications. We developed the app *Delay Explore* to conduct the real-time measurements on a mobile phone, considering both fixed and mobility scenarios in an LTE-based MEC network. The app sends ping streams to the EN and CN and collects the feedback as RTT data. In addition, our app can also collect other parameters, such as signal strength indicators, vehicle speed, and location information. Based on observed RTTs, the data is split into

[2]For simplicity, we use the term 'packet size' to refer to the size of the data associated with a given task.

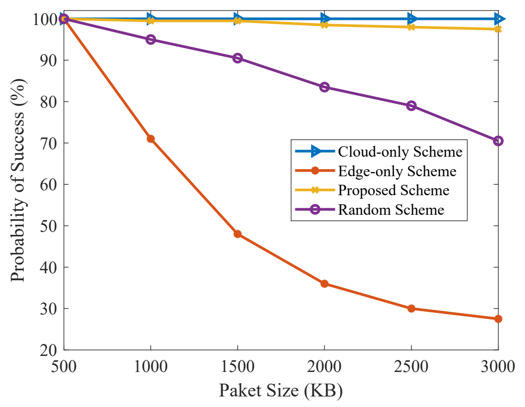


Fig. 20. Probability of success vs. packet size under different offloading schemes.

the trend and the residual by appropriate filtering. We first proposed a K-medoids clustered LSTM network to predict the trend part, assisted by RSRP, RSRQ, and SS. We then combined Epanechnikov Kernel function-based probabilistic sampling with the average function to predict the residual part. Performance tested on real-time collected data shows that the proposed integration approach can achieve lower prediction error and higher stability than sampling- and filtering-based predictors.

After obtaining the prediction of the propagation delay, we extended our work by simulating the transmission, queueing, and processing delays and accounted for the E2E delay. The E2E delay is analyzed under different settings of the server, including the task size, task complexity, and transmission/computation rates. We also considered the impact of the buffer sizes of CN/EN, where tasks that arrive at the full buffer are discarded and are denoted as failed transmissions. We then designed an offloading algorithm that allows a vehicular user to select the cloud/edge server and reduce the E2E delay based on the estimated E2E delay. The simulation shows that the proposed approach can reduce the average E2E delay by around 60% compared with a random node selection offloading scheme while maintaining the probability of success of more than 95% even when the packet size is large.

References

[1] W. Zhang, M. Feng, M. Krunz, and H. Volos, “Latency Prediction for Delay-sensitive V2X Applications in Mobile Cloud/Edge Computing Systems,” in *Proc. IEEE GLOBECOM’20*, pp. 1-6, Dec. 2020

[2] NGMN Alliance, “V2X white paper,” June 2018.

[3] D. Feng, C. Haase-Schütz, L. Rosenbaum, H. Hertlein, C. Glaeser, F. Timm, W. Wiesbeck, and K. Dietmayer, “Deep multi-modal object detection and semantic segmentation for autonomous driving: Datasets, methods, and challenges,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 3, pp. 1341-1360, Mar. 2021.

[4] M. Wu, F. R. Yu, and P. X. Liu, “Intelligence networking for autonomous driving in beyond 5G networks with multi-access edge computing,” *IEEE Transactions on Vehicular Technology*, vol. 71, no. 6, pp. 5853-5866, June 2022.

[5] 5GAA, “Toward fully connected vehicles: Edge computing for advanced automotive communications,” White Paper, Dec. 2017.

[6] 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.

[7] Y. Xiao and M. Krunz, “AdaptiveFog: A modelling and optimization framework for fog computing in intelligent transportation systems,” *IEEE Transactions on Mobile Computing*, pp. 1-15, May 2021.

[8] M. K. Abdel-Aziz, S. Samarakoon, C. Liu, M. Bennis, and W. Saad, “Optimized age of information tail for ultra-reliable low-latency communications in vehicular networks,” *IEEE Transactions on Communications*, vol. 68, no. 3, pp. 1911-1924, March 2020.

[9] C. -L. Chen, C. G. Brinton, and V. Aggarwal, “Latency minimization for mobile edge computing networks,” *IEEE Transactions on Mobile Computing*, pp. 1-15, Oct. 2021.

[10] Y. Liu, H. Yu, S. Xie, and Y. Zhang, “Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks,” *IEEE Transactions on Vehicular Technology*, vol. 68, no. 11, pp. 11158-11168, Nov. 2019.

[11] 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.

[12] M. Feng, M. Krunz, and W. Zhang, “Task partitioning and user association for latency minimization in mobile edge computing networks,” in *Proc. of the IEEE Conference on Computer Communications Workshops*, 2021, pp. 1-6

[13] Z. Hou, C. She, Y. Li, L. Zhuo, and B. Vucetic, “Prediction and communication co-design for ultra-reliable and low-latency communications,” *IEEE Transactions on Wireless Communications*, vol. 19, no. 2, pp. 1196-1209, Feb. 2020

[14] R. Zhu, B. Liu, D. Niu, Z. Li, and H. V. Zhao, “Network latency estimation for personal devices: A matrix completion approach,” *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 724-737, April 2017.

[15] R. Tripathi and K. Rajawat, “Adaptive network latency prediction from noisy measurements,” *IEEE Trans. Netw. Service Manag.*, doi: 10.1109/TNSM.2021.3051736.

[16] 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.

[17] X. Wang, Z. Yu, and S. Mao, “Indoor localization using magnetic and light sensors with smartphones: A deep LSTM approach,” *Mobile Networks and Applications Journal*, vol.25, pp. 819–832, Apr. 2020.

[18] J. Cheng, Y. Liu, Q. Ye, H. Du, and A. V. Vasilakos, “DISCS: A distributed coordinate system based on robust nonnegative matrix completion,” *IEEE/ACM Transactions on Networking*, vol. 25, no. 2, pp. 934-947, Apr. 2017.

[19] M. Feng, M. Krunz, and W. Zhang, “Joint task partitioning and user association for latency minimization in mobile edge computing networks,” *IEEE Transactions on Vehicular Technology*, vol. 70, no. 8, pp. 8108-8121, Aug. 2021

[20] M. Mudassar, Y. Zhai, and L. Lejian, "Adaptive fault-tolerant strategy for latency-aware IoT application executing in edge computing environment," in *IEEE Internet of Things Journal*, vol. 9, no. 15, pp. 13250-13262, Aug. 2022.

[21] B. Kopras, B. Bossy, F. Idzikowski, P. Kryszkiewicz, and H. Bogucka, "Task allocation for energy optimization in fog computing networks with latency constraints," in *IEEE Transactions on Communications*, pp.1-15, Oct. 2022.

[22] S. Yuan, J. Li, H. Chen, Z. Han, C. Wu, and Y. Zhang, "JIRA: Joint incentive design and resource allocation for edge-based real-time video streaming systems," in *IEEE Transactions on Wireless Communications*, pp.1-15, Oct. 2022.

[23] 3GPP, “3GPP radio resource control (RRC) (Release 10),” 3GPP PS 36.331, v10.14.0, Sep 2014.

[24] I. Hadžić, Y. Abe, and H. C. Woithe. “Edge computing in the ePC: a reality check,” *Proc. of ACM/IEEE Symposium on Edge Computing*, pp. 1–10, New York, NY, USA, 2017.

[25] D. Renga, Z. Umar, and M. Meo, “Trading off delay and energy saving through advanced sleep modes in 5G RANs,” *IEEE Transactions on Wireless Communications*, pp. 1-12, March 01, 2023.

[26] M. Feng, S. Mao, and T. Jiang, “Base station ON-OFF switching in 5G wireless networks: Approaches and challenges,” *IEEE Wireless Communications*, vol. 24, no. 4, pp. 46-54, Aug. 2017.

[27] K. Chen and L. Huang, “Timely-throughput optimal scheduling with prediction,” *in IEEE/ACM Trans. Netw.*, vol. 26, no. 6, pp. 2457-2470, Dec. 2018.

[28] H. Volos, T. Bando, and K. Konishi, “Latency modeling for mobile edge computing using LTE measurements,” *IEEE VTC’18*, Chicago, IL, pp. 1–5.

[29] E. Schubert and P. J. Rousseeuw. “Faster k-Medoids clustering: improving the PAM, CLARA, and CLARANS algorithms.” in *Proc. In-*

ternational Conference on Similarity Search and Applications, Springer, Cham, pp. 171-187, 2019.

[30] F. O. Olowononi, D. B. Rawat, and C. Liu, "Resilient machine learning for networked cyber physical systems: A survey for machine learning security to securing machine learning for CPS," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 1, pp. 524-552, Nov.9 2020.

[31] L.J.P Van Der Maaten, and G. E. Hinton, "Visualizing data using t-SNE," *Journal of machine learning research*, vol. 9, no. 11, pp. 2579-2605, 2008.

[32] P. J. Rousseeuw, "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *Journal of Computational and Applied Mathematics*, vol. 20, pp. 53-65, 1987.

[33] D. L. Davies and D. W. Bouldin, "A Cluster Separation Measure," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 224-227, April 1979.

[34] I. Pelle, F. Paolucci, B. Sonkoly, and F. Cugini, "Latency-sensitive edge/cloud serverless dynamic deployment over telemetry-based packet-optical network," *IEEE Journal on Selected Areas in Communications*, vol. 39, no. 9, pp. 2849-2863, Sept. 2021.

[35] F. Yang, M. Tang, and O. Sinanoglu, "Stripped functionality logic locking with hamming distance-based restore unit (SFLL-hd) – unlocked," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 10, pp. 2778-2786, Oct. 2019.

[36] C. Yardim, P. Gerstoft, and W. S. Hodgkiss, "Tracking refractivity from clutter using Kalman and particle filters," *IEEE Transactions on Antennas and Propagation*, vol. 56, no. 4, pp. 1058-1070, Apr. 2008.

[37] S. Sharma, A. Majumdar, V. Elvira, and É. Chouzenoux, ""Blind Kalman filtering for short-term load forecasting," *IEEE Transactions on Power Systems*, vol. 35, no. 6, pp. 4916-4919, Nov. 2020.

[38] S. Yi and M. Zorzi, "Robust Kalman filtering under model uncertainty: The case of degenerate densities," *IEEE Transactions on Automatic Control*, vol. 67, no. 7, pp. 3458-3471, Jul. 2022.

[39] M. S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing*, vol. 50, no. 2, pp. 174-188, Feb. 2002.

[40] A. Aspeel, A. Gouverneur, R. M. Jungers, and B. Macq, "Optimal intermittent particle filter," *IEEE Transactions on Signal Processing*, vol. 70, pp. 2814-2825, Jun. 2022.

[41] W. Song, Z. Wang, J. Wang, F. E. Alsaadi, and J. Shan, "Distributed auxiliary particle filtering with diffusion strategy for target tracking: A dynamic event-triggered approach," *IEEE Transactions on Signal Processing*, vol. 69, pp. 328-340, Dec. 2020.

Wenhan Zhang [S'19] received the B.S. degree in electrical engineering and automation from Hefei University of Technology, China, in 2016, and the M.S. degree in electrical engineering from Syracuse University in 2018. He is working toward his Ph.D. degree with the Department of Electrical and Computer Engineering at the University of Arizona. His research interests include mobile edge computing, wireless communications, and applications of machine learning in wireless networks.

Mingjie Feng [S'15] is currently a Full Professor with the Research Center of 6G Mobile Communications, Wuhan National Laboratory for Optoelectronics, Huazhong University of Science and Technology, Wuhan, China. He is a recipient of the Best Paper Award of Digital Communications and Networks, the Woltosz Fellowship from Auburn University, and the Best Reviewer of IEEE Transactions on Wireless Communications. He has served/is serving as an Associate Editor for several journals in communications, including IEEE Networking Letters and Digital Communications and Networks. He was/is a Technical Program Committee Member of various IEEE conferences, including IEEE MASS and IEEE ICC.

Marwan Krunz [S'93-M'95-SM'04-F'10] is a Regents Professor of electrical and computer engineering at the University of Arizona. He also holds a joint appointment as a professor of computer science. From 2015 to 2023, he was the Kenneth VonBehren Endowed Professor in ECE. Currently, he directs the Broadband Wireless Access and Applications Center (BWAC), a multi-university NSF/industry center that focuses on next-generation wireless technologies. He is also an Affiliated Faculty of the UA Cancer Center. Previously, he served as the Site Director for the Connection One Center. He served as the chief scientist for two startup companies that focus on 5G and beyond systems and machine learning for wireless communications. He has published more than 330 journal articles and peer-reviewed conference papers and is a named inventor on ten patents. His latest H-index is 62. His research interests include wireless communications and protocols, network security, and machine learning. He was an Arizona Engineering Faculty Fellow and an IEEE Communications Society Distinguished Lecturer. He received the NSF CAREER Award. He was the TPC Chair for several conferences and symposia, including INFOCOM'04, SECON'05, WoWMoM'06, and Hot Interconnects 9. He was a general chair for WiOpt'23, vice-chair for WiOpt'16, and the general co-chair for WiSec'12. He served as the Editor-in-Chief for the IEEE Transactions on Mobile Computing. He served as an editor for numerous IEEE journals.