BOSTON COLLEGE

SENIOR THESIS

---

# Assisting Federated Learning with Vehicular Clouds

---


*Author:*

Yicheng Shen

*Supervisor:*

Lewis Tseng



May 21, 2021

# *Abstract*

Department of Computer Science

**Assisting Federated Learning with Vehicular Clouds**

by Yicheng Shen

In recent years, Federated Learning (FL) emerged as a machine learning technique that avoids the step of centralizing local data from edge devices in a data center. Collecting only the gradients computed from local on-device data, FL enables edge devices to upload less bytes during communication and protects users' privacy. Moreover, FL algorithms are suitable for many use cases, such as recommendation systems, decision making, risk control, and pattern recognition.

This thesis aims to integrate vehicular clouds with FL in order to improve the performance of FL. Smart vehicles nowadays are equipped with more and more computing power. As their on-board computers do not operate at full-speed at all times, we can use their excess computing power to perform training. We present a system named VC-SGD (Vehicular Clouds-Stochastic Gradient Descent) that supports FL by using vehicular clouds and edge devices. Our three-layer architecture consists of a cloud server, some edge servers which are deployed with vehicular clouds, road-side units, base stations, etc., and many edge devices which are workers. In addition, this thesis investigates the data collection problem by simulating real-time location-specific data. We implement a simulator which utilizes SUMO to simulate vehicle mobility and MXNet to perform machine learning tasks. Finally, we used the simulator to evaluate the performance of VC-SGD.

# *Acknowledgements*

I would like to express my gratitude to my supervisor, Lewis Tseng, who has been patient in guiding my entire research process and gave me countless wise suggestions when I encountered obstacles. My research and thesis would not have been possible without the support from Professor Tseng. I also would like to thank my great research teammate, Anran Du, who is smart and dedicated. I will remember the numerous times we worked together over Zoom and will be glad to collaborate with you again in the future.

# Contents

# Chapter 1

# Introduction

## 1.1 Background

In recent years, an increasing number of machine learning algorithms are being utilized for various applications. These emerging applications serve many different important functionalities, such as decision making in smart environments and ambient intelligence [2], pattern recognition as in object and scene understanding in augmented reality [13], and face recognition in real-time identity verification. The focus of this thesis is on a significant family of machine learning algorithms, *federated learning* (FL). More specifically, we aim to assist FL by using vehicular clouds.

FL was introduced by Google [5]. FL is different from standard machine learning due to its decentralized nature. Standard machine learning requires data to be centralized in one single machine or data center. In contrast, FL avoids the step of sending local data on edge devices to central servers. Instead of sending data to central servers, edge devices send gradients computed from local data to central servers to update the model. This approach is special since it allows training machine learning models without sharing data, thus providing a promising feature that preserves users' privacy. From users' perspective, their data are not shared with third parties and the training is performed only locally on their devices.

In the field of edge-based FL systems, one aspect that needs more investigation is using real-time location-specific data to train a machine learning model [17, 12, 14]. This problem is challenging because of two factors. Firstly, it requires a real-time data stream classifier that is adaptive to concept drift (change of patterns and hence, the machine learning model). Besides, the data is location-specific in the sense that data collected from different locations contain different unique characteristics. Such an *online* training is important for pervasive systems, as the characteristics and patterns in such systems tend to change over time and evolve due to varying environments and/or user interactions. However, the existing systems, e.g., [9, 15, 8], do not provide a thorough and satisfying solution (as detailed in Section 2).

## 1.2 Main Contributions

We address the problem by proposing a novel system, named VC-SGD, which integrates edge-based FL and vehicular clouds. Looking at our system from a high-level, we use a vehicular cloud as a virtual edge server. In VC-SGD, a vehicular cloud is formed by a cluster of nearby vehicles; a virtual edge server serves functionalities of collecting information regarding data at a specific location, performing aggregation of gradients, and communicating with the cloud server and edge devices.

VC-SGD is an effective system because the virtual edge servers enable users to collect important location-based data for the real-time training of machine learning models. When physical edge servers are not available, the virtual edge servers which are enabled by the vehicular clouds in VC-SGD will serve as backups, thus improving the availability of edge servers. Without vehicular clouds, it will be difficult to collect information in some areas that do not have a physical edge server. In extreme cases, some location-specific data will never be trained and used to update the model. In VC-SGD,

virtual edge servers powered by vehicular clouds serve as a solution to this problem.

According to the architecture of VC-SGD, we build a simulator for both the investigation of the problem related to real-time location-specific data and the evaluation of our system's performance. The simulator combines vehicle mobility simulation and machine learning training. It utilizes SUMO to simulate continuous traffic flows on a real world map and uses MXNet to run machine learning.

We also investigate the "data collection" problem, specifically for collecting and using real-time and location-specific data. This is implemented as a feature of the simulator: to achieve the real-time location-specific characteristics, the data are partitioned based on classes and the partitions are placed on different regions in the map. This feature will be explained in more details in Section 4.1.

Lastly, we evaluate the performance of VC-SGD. The evaluation results obtained from experiments ran on our simulator demonstrate that VC-SGD helps improve both accuracy and efficiency.

# Chapter 2

# Preliminaries

In standard edge computing frameworks, there are notions of edge devices, edge servers, and central servers. We consider the standard three-level architecture [17, 12]. The bottom level consists of the edge devices (e.g., IoT devices, sensor nodes, mobile phones, high-end vehicles, etc.) used by clients or end users. The middle layer is the "edge or fog computing layer," which consists of the edge servers in the form of proxy servers, routers, road-side units, cloudlets, or base stations, etc. These nodes are capable of providing essential storage, communication, and computation capabilities. The top level is the cloud computing layer, which consists of the cloud data center(s) and servers.

## 2.1 Problem Formulation

Our target scenario focuses on pervasive applications that use real-time location-specific data. Particularly, users may have distinct behaviors given their location. In the system, each edge device collects location-specific data at its current location in a real-time fashion, and all the nodes, including servers, jointly aim to learn an online machine learning model using a FL algorithm. During the training process, users' data is trained locally and is never transmitted out of the device; hence, data privacy is preserved.

*Example applications:* Object detection and identification is essential for many pervasive systems such as augmented reality-based applications and intelligent transportation systems for improving road safety.

*Data Collection Problem:* Most of the existing systems [17, 12, 14] assume that the datasets are already collected and did not consider how to collect real-time and location-specific data so that the ML/FL algorithms can be trained efficiently. Concretely, we are interested in cost-effective approaches for supporting efficient data collection.

*Limitation of Existing Solutions:* The lack of study on the "data collection" problem limits the applicability of edge-based FL to pervasive systems. In the original use case of FL algorithms [5], the training data is specific to each mobile phone (or more precisely the phone owner's data); hence, the phone can eventually obtain all the necessary data for training a machine learning model with a desirable accuracy. However, in the scenarios with real-time location-specific data, the original system in [5] faces a serious issue: the machine learning model trained using an FL approach can never achieve a satisfying accuracy and/or speed in the case that the information regarding some important data is never used for training. This situation is possible based on the following observations:

- Training a satisfying ML model for various purposes, including object detection and identification applications, requires a huge amount of diverse and comprehensive image data, e.g., distinct objects from different angles and lighting.

- On one hand, typical edge devices are not powerful enough to store a complete history of high-resolution images or data collected for a long period of time. Hence, by the design of the three-level edge-based FL, only data near edge servers is used for training. On the other hand,

cloud servers will be the bottleneck [15, 8] if the devices upload all the collected data due to the large size of data.

- Important information can be captured by sensors on edge devices from anywhere and at any moment. It is impossible to predict which data or data at which location is important in advance; hence, it is unrealistic to deploy edge servers to cover all the "important data."

## 2.2 Vehicular Clouds

As vehicles become more and more powerful, they can be used as resources to run machine learning tasks. Nowadays, many smart vehicles are equipped with on-board units with decent computing power. When the on-board units are idle or not running at full capacity, we can make use of these resources by assigning edge computing tasks to them. In this setting, the integration of vehicular clouds will be suitable and effective.

Vehicular clouds (VC) [4, 11] is an emerging research direction that investigates the synergy in the concept of cloud/edge computing and connected vehicles. The key purpose is to provide "cloud services" such as communication, storage, and computation, by leveraging resources of the on-board units on high-end and connected vehicles. Recent works have proposed to use vehicular clouds for various intensive computation services, such as MapReduce computation, [6], [1]. To our knowledge, no prior work has investigated the integration of VC with edge-based FL.

# Chapter 3

# VC-SGD

We propose our framework, named VC-SGD, which integrates FL and vehicular clouds. VC-SGD supports a general family of FL algorithms, stochastic gradient descent (SGD). Prior works [9, 15, 8] have studied SGD in various settings in the edge computing paradigm, but as we have discussed in Section 2, they did not consider the data collection problem for real-time location-specific data. We attempt to address and tackle this issue in our design of VC-SGD. In this section, we will show the architecture and algorithm of VC-SGD.

## 3.1 Architecture

VC-SGD has a three-layer architecture shown in Figure 3.1.

- The top layer consists of a cloud server. It sends the most up-to-date parameters of the machine learning model to edge servers and collects computed gradients sent from edge servers. In addition, when enough gradients are collected from edge servers, it updates the machine learning model.

- The middle layer consists of two types of edge servers: physical ones and virtual ones. The physical edge servers are infrastructures, such as

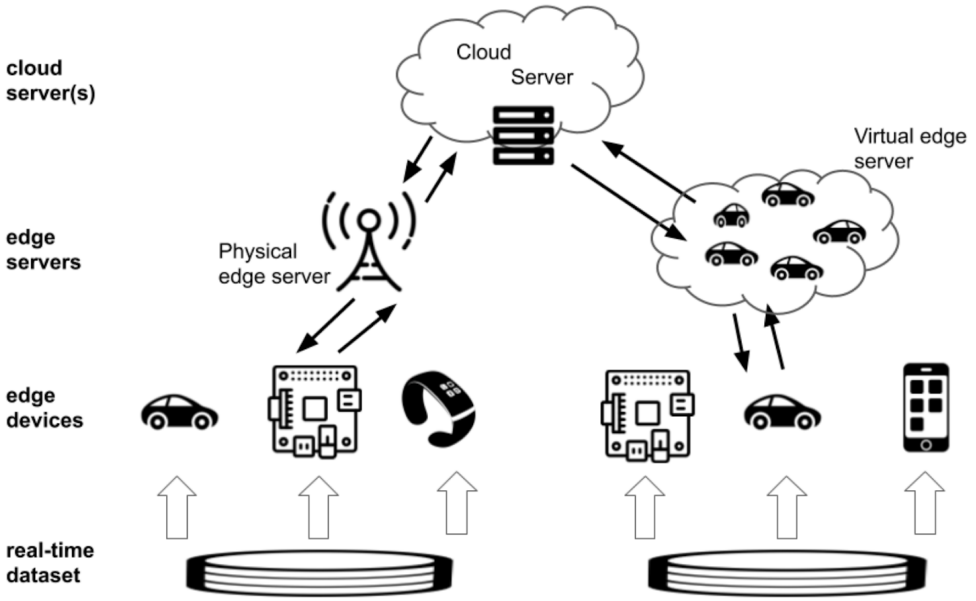


FIGURE 3.1: The architecture of VC-SGD. The stationary vehicular clouds enable the deployment of virtual edge servers. By assumption, each location potentially has a unique dataset; hence, in the figure, devices near two edge servers in different locations collect data from different real-time datasets.

roadside units, base stations, etc. The virtual edge servers are the vehicular clouds, which are formed by multiple nearby vehicles. The edge servers periodically receive parameters of the model from the cloud server. Then, they are responsible for sending parameters of the model to edge devices. Moreover, they need to collect and aggregate gradients uploaded by the edge devices. After each aggregation, they upload the aggregated gradients to the cloud server.

- The bottom layer consists of edge devices, such as mobile devices, on-board units equipped on vehicles, etc. The tasks for the edge devices are collecting real-time location-specific data, running machine learning training on the data, and uploading gradients to edge servers.

Our design integrates vehicular clouds as virtual edge servers in the middle layer. This enhancement is the key difference from the standard three-level edge-based FL. In the integration of vehicular clouds, we make an essential design choice to focus on the stationary vehicular cloud. That is, the position of a vehicular cloud is fixed. According to the previous work [7], this type of vehicular cloud is able to provide more stable storage and communication capacity, in dealing with localized data, than the dynamic vehicular cloud, i.e., which moves with a cluster of vehicles.

Because a vehicular cloud is formed by a cluster of nearby vehicles, in the case that not enough number of vehicles is around, a stationary vehicular cloud might disappear. SGD is inherently robust to tolerate missing information from disappeared vehicular clouds in this case.

Stationary vehicular clouds are designed to be at locations where many vehicles frequently stop at. For example, near intersections or parking lots, vehicles stay still for a period of time before they depart and stationary vehicular clouds can be formed by these vehicles. A typical stationary vehicular cloud is able to cover a range around an intersection and handle edge devices within its range. Therefore, VC-SGD in fact needs to utilize many vehicular clouds to support the training of FL algorithms efficiently.

The main benefit of utilizing stationary vehicular clouds is that we are able to provide more "coverage" with a minimal change to the existing edge-based FL. As long as the edge devices has the capability to communicate with the connected vehicles (inside a vehicular cloud), they can run essentially the same FL algorithm; otherwise, a simple relay mechanism is necessary. In short, in most cases, edge devices can behave the same regardless of communicating with a physical or virtual edge server. This modular design enables VC-SGD to use a general set of FL algorithms, e.g., [17], [12]. In the rest of the thesis, we will refer to the stationary vehicular clouds as virtual edge servers.

The original FL system in [5] does not have the layer of edge servers;

however, as identified in prior works [17, 12], this cloud-based solution is not scalable. Prior solutions [9, 15, 8] used edge servers to improve scalability, but they assumed that the physical edge servers are able to collect all the important data. However, the collection of the important data has not been investigated. Therefore, VC-SGD complements prior systems by addressing the "data collection" problem of real-time location-specific data.

## 3.2 Algorithm Preliminary

As the first step, VC-SGD supports a widely adopted training mechanism – distributed Stochastic Gradient Descent (SGD) algorithms – which are popular in the optimization and machine learning literature, e.g., [3]. Given a cost function $Q$, the SGD algorithm is designed to output an optimal parameter $\theta^*$ in a $d$-dimensional space such that the cost function is minimized. An SGD algorithm executes in an iterative fashion, where in each round $t$, the algorithm computes the gradient of the cost function $Q$ at parameter $\theta^t$ and updates the parameter using the gradient descent approach.

Each physical edge server or virtual edge server $j$ has a local cost function $Q_j$, which captures the characteristics of the real-time location-specific data. That is, each distinct location has a different set of data pattern. The distributed SGD algorithm aims to minimize the overall cost function and compute $\theta^*$ such that the sum of all the cost functions is minimized:

$$\theta^* = \underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{j=1}^{n} Q_j(\theta). \tag{3.1}$$

A popular distributed framework to parallelize the computation is the *parameter server model*, in which the parameter server distributes the computation tasks to workers and aggregates their computed gradients to update the parameters (of the machine learning model) in each round [16]. In edge

computing, it is difficult for a single cloud to manage all the devices; hence, various edge-based training paradigms have been proposed [17, 12].

In our three-layer edge computing architecture (as shown in Figure 3.1), both (virtual) edge servers and the cloud server need to perform the aggregation, but only the cloud server will update the parameters of the model. In the usage scenario, each gradient computation is over a small batch of data samples; hence, the computation is very efficient, even on a small device. Moreover, there is a trend for deploying highly specialized chips for fast and energy-efficient deep learning tasks, e.g., TPU (Tensor Processing Unit) from Google. We believe that in the near future, edge devices will be fully capable of performing such a gradient computation.

For brevity of presentation, we assume a synchronous system, i.e., each worker and server proceed in a lock step. We stress that it is simple to generalize the assumption about synchrony.

## 3.3 Algorithm Framework

We present a framework for integrating a distributed SGD algorithm with our three-layer architecture. In the following workflow, all the servers and edge nodes proceed in synchronous rounds, and execute the following steps in each round $t \geq 0$:

1. Cloud server sends parameter $\theta^t$ to all edge servers.
2. Edge server $j$ forwards $\theta^t$ to nearby workers $w_{j,k}^t$.
3. Each worker $w_{j,k}^t$ randomly chooses a (small) data batch from its local dataset that is collected using on-device sensors from its current location, and computes a local estimate gradient $g_{j,k}^t$.
4. Worker then sends $g_{j,k}^t$ back to the edge server $j$.

5. Upon collecting enough gradients from nearby workers, edge server $j$ computes and sends an aggregated gradient $g_j^t$ to cloud server.

6. Upon collecting aggregated gradients from all the edge servers, cloud server then updates the parameter using the gradient descent approach with step size $\eta$:

$$\theta^{t+1} = \theta^t - \eta \sum_{j=1}^{n} g_j^t.$$

The framework is general in the sense that prior algorithms [15, 8] can be easily integrated by replacing the needed parameters and aggregation rules. If the FL algorithms are robust to stale data, then at step 3, we can make the convergence faster with a simple modification: if the worker has enough storage space, then the worker may also choose to use some (stale) data collected at other places to form a batch for training in this round. In our preliminary design, if the virtual edge server does not have enough vehicles inside its range, then the computed gradients will be discarded. Most FL algorithms are robust to this type of intermittent data loss.

# Chapter 4

# Evaluation

## 4.1 Simulator

We build a simulator to help investigate the collection of real-time location-specific data and evaluate the performance of VC-SGD. Our simulator uses Simulation of Urban Mobility (SUMO), which is a continuous traffic simulation package designed to handle large networks, for simulating the mobility pattern of the edge devices and vehicles. It also uses the MXNet framework for real machine learning training.

As mentioned in our problem formulation in Section 2, we need to ensure two important features of data collection: the data should be collected in a *real-time* and *location-specific* fashion in our simulation. To achieve these two features, we partition the map in our simulation. The map shown in Figure 4.1 is generated by SUMO OSMWeb Wizard and is based on the actual map of an area around the Boston Common located in downtown Boston. We visualize the traffic data on the map by plotting traces of vehicles as dots and many connected dots form the contour of the roads.

The map is partitioned into 10 regions shown in different colors by using a clustering algorithm and an algorithm of Voronoi diagram. The partition is based on historical traffic data. First, we run a density-based clustering algorithm, named DBSCAN, on the traffic data generated by SUMO. We select 10 clusters with the most traffic and find their centers which are shown as black

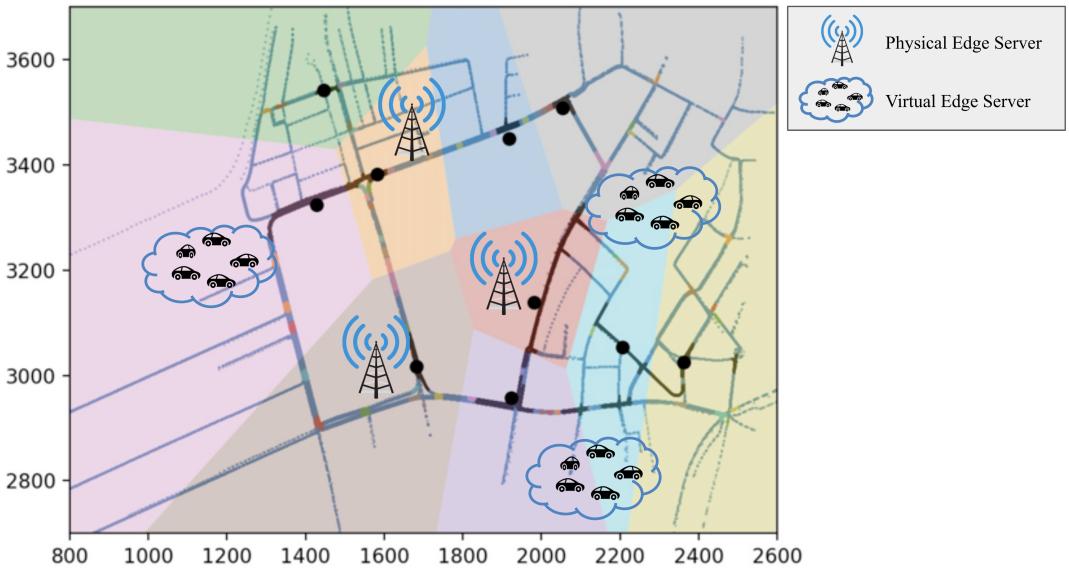


FIGURE 4.1: Illustration of Partitioned Map with Zone Data.

dots in Figure 4.1. We consider these 10 center points as seeds in Voronoi diagram and generate the partitioned map using Euclidean distance as the metric.

In each distinct partition on the map, the data will be different. We achieve this by processing the dataset and assigning each partition with different batches of data before training starts in our simulation. Therefore, the location-specific feature of data is established. During the simulation, when an edge device travels through one of the partitions, it will obtain a batch of data. This simulates the edge device's sensors gathering information from its surroundings, thus establishing the real-time feature of data.

In our experiment, we place both physical and virtual edge servers at pre-selected locations with high traffic flows based on historical traffic data. Note that the icons of edge servers shown in Figure 4.1 do not represent their actual selected locations. They are used to demonstrate that physical edge servers are not present in every partition of the map. This simulates the real circumstance where the infrastructures do not have full coverage in remote areas. The virtual edge servers play important roles in the regions that are not covered by physical edge servers.

## 4.2 Experiment Setup

We first design an experiment that runs on our simulator to verify that our design can effectively handle primitive applications. One common theme of the advanced machine learning algorithms used in pervasive applications is image recognition. If one cannot successfully solve the image recognition task, then the more advanced tasks such as object detection, object recognition, and image segmentation are close to impossible to be solved.

For our preliminary evaluation, we start with the popular benchmark dataset, MNIST handwritten digits [10], which contains 60k training examples and 10k testing examples. There are 10 classes, which are the digits from 0 to 9, in the dataset. The task for edge devices is to train on examples in the dataset and collectively improve the model for recognition of handwritten digits.

Our machine learning model is a Sequential model with 3 dense layers. The learning rate is set to $5 * 10^{-4}$. We use top-1 accuracy on the testing set and the cross-entropy loss function on the training set as the evaluation metrics. The accuracy is evaluated by comparing the prediction made by our trained machine learning model with the test set.

Before we start our experiment and the training, we process the MNIST dataset to simulate the *real-time* and *location-specific* features by doing the following steps:

1. Classify training examples into 10 sets according to their classes (0-9).
2. From each set, reserve $x$% of examples as "zone data" and take out the remaining $(100 - x)$% of examples.
3. Mix the examples taken out from all 10 sets together for usages as mixed data.
4. Assign $\frac{1}{10}$ of the mixed data to each set.

5. For each set, assign the training examples in the set to one of the partitions on the map.

6. In each set, shuffle and batch the examples prior to the start of each round.

In step 2, "zone data" means that the data examples are all from one class. In terms of the MNIST dataset, "zone data" means labels of the data are the same digit. In step 4, after evenly distributing the mixed data to every set, each set has a portion of "zone data" and a portion of mixed data. After step 6, each partition uses a queue to contain the batched training examples and an edge device will fetch a batch as it travels through one of the partitions.

After steps of processing, the MNIST data stream (collected by each edge device) is non-i.i.d. or "disjoint local." That is, data batches at each queue draw from a slightly different distribution. Concretely, a queue is configured to contain $x$% of data from the "zone data" (data from a pre-assigned class) and $(100 - x)$% of mixed data from the remaining data in the dataset.

| **Experiment Parameters** | |
|---|---|
| Number of vehicles | 647 |
| Peak number of vehicles | 43 |
| Duration | 3864 seconds |
| Edge server range | 100 meters |
| Minimum number of vehicles in a VC | 5 |
| Number of physical edge servers (RSU) | 5 |
| Number of virtual edge servers (VC) | 0, 3, or 6 |
| Batch size | 100 |
| Number of rounds | 100 |
| Number of accumulative gradients | 10 |

TABLE 4.1: Parameters used in the experiment.

Table 4.1 shows the parameters used in our experiment. The vehicle mobility simulated by SUMO generates a total of 647 vehicles traveling through the map. The duration from the first vehicle entering the map to the last vehicle leaving the map is 3,864 seconds. In the peak second that has the most

traffic, there are 43 vehicles on the map simultaneously. When all the vehicles exit the map, we start the traffic flow over from the beginning in order to guarantee continuous flow of vehicles during our experiment. We assume the edge server's communication range has a radius of 100 meters. Edge devices can only communicate with edge servers within edge servers' range. We also set the minimum number of vehicles to form a vehicular cloud at the pre-selected location to be 5.

The batch size is set to 100, meaning that each edge device obtains a batch of 100 location-specific data examples as it enters a partition. The number of accumulative gradients is 10, which suggests that the cloud server and the edge servers need to wait for 10 gradients to perform an aggregation. We run training for 100 rounds; exhausting all data batches in all partitions on the map means completion of a round. Prior to a new round, data will be shuffled and batched again.

## 4.3 Experimental Results

The evaluation runs on the Google Cloud Platform (GCP). The machine type is e2-standard-4, which has 4 vCPUs and 16GB memory. The system is ubuntu-1804-bionic-v20210514. We run experiments according to our experiment setup and report the average results of 5 runs for each set of parameters. Figure 4.2 presents the results. We test 3 different combinations of RSUs (physical edge servers) and VCs. In all 3 cases, the number of RSUs is set to 5; we vary the number of VCs to evaluate the effect of introducing different number of VCs.

With more VCs, the model collectively trained by edge devices achieves a higher accuracy. Moreover, with more "zone data" – i.e., each zone has a higher proportion unique data – our framework VC-SGD performs better, in terms of both accuracy and convergence speed. This is because with the help

from VCs, edge devices have a higher chance to train with the real-time data they collect and upload their computed gradients.

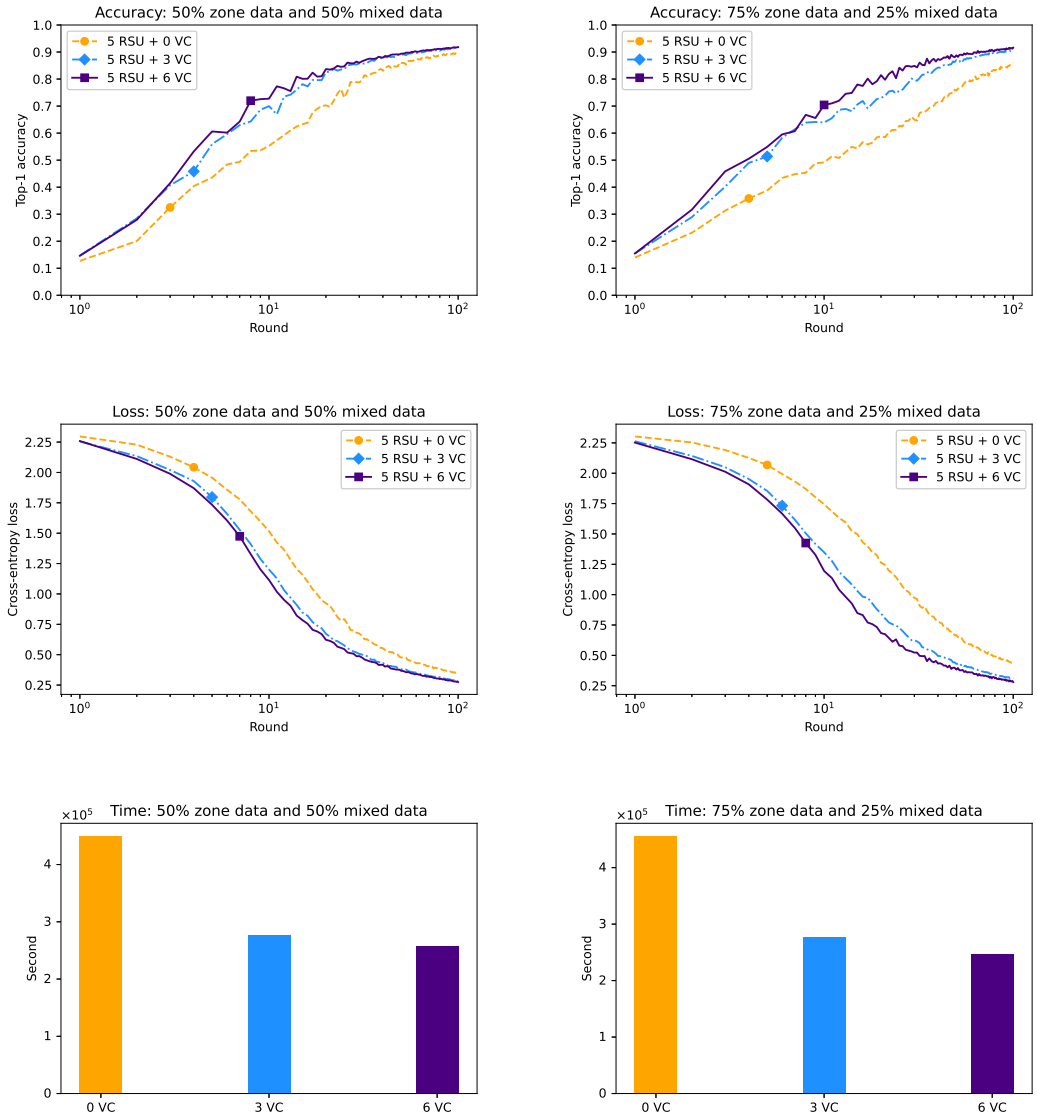


FIGURE 4.2: Results of Mean-based FL with 50% Zone data (Left column); Results of Mean-based FL with 75% Zone data (Right column).
Plots of Top-1 accuracy (First row); Plots of Cross-entropy loss (Second row); Plots of total time (number of timesteps in SUMO) used to complete 100 rounds (Third row).

In the cases with more VCs, the time used to complete 100 rounds is shorter, as shown in the third row of Figure 4.2, because more available VCs can enable more edge devices to contribute to the training in the same

amount of time. When no VC is used, edge devices can only upload gradients when they pass by the physical edge servers. Under this circumstance, many edge devices cannot train their local data and upload gradients because of the lack of edge servers. In our simulation, they cannot collect new data if they do not complete training their existing local data. So, each round takes much longer when the number of VCs is 0. In contrast, with VCs, edge devices can gather and train new data more frequently because they are more likely to pass by edge servers. Therefore, introducing more VCs help improve performance and efficiency.

Recall that edge devices will discard data if they are not near any physical or virtual edge server. This is a reasonable assumption adopted in the literature [17, 12], because (i) the number of edge devices will become too large to be handled directly by the cloud server; and (ii) the edge devices do not typically have a large enough space to store prior data. This is why we believe that it is important to use the concept of vehicular virtual edge server to address the data collection problem, and enable FL algorithms for pervasive systems.

# Chapter 5

# Summary and Future Work

We propose VC-SGD to assist FL with vehicular clouds, and use a customized simulator to evaluate its efficacy. VC-SGD addresses the missing link in the literature – collecting and training with real-time location-specific data.

VC-SGD can be extended to more future directions and we can investigate more about the following topics:

- *Extensive evaluation:* we have only implemented a mean-based FL into our simulator; we plan to extend our simulator to more FL algorithms and study practical trade-offs such as communication cost, different synchrony assumption, and the impact of traffic pattern.
- *Advanced problems and integration with pervasive systems:* we will study the efficacy of FL algorithms for more advanced problems such as object detection, and integrate the results with pervasive systems.
- *Robustness:* failures, or even attacks, are bound to happen in edge-base FL systems. Making VC-SGD more robust is an important challenge.
- *Explorations of deployment on actual vehicles:* VC-SGD is currently run on our simulator, but it will be important to evaluate its performance on vehicles in reality; the amount of computing power it requires and its energy consumption will be future problems to consider.

# Bibliography

[1] Samiur Arif et al. "Datacenter at the Airport: Reasoning about Time-Dependent Parking Lot Occupancy". In: *Parallel and Distributed Systems, IEEE Transactions on* 23 (Nov. 2012). DOI: 10.1109/TPDS.2012.47.

[2] Juan Augusto Wrede. "Ambient Intelligence: The Confluence of Ubiquitous/Pervasive Computing and Artificial Intelligence". In: *Intelligent Computing Everywhere* (Jan. 2007). DOI: 10.1007/978-1-84628-943-9_11.

[3] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511804441.

[4] Mohamed Eltoweissy, Stephan Olariu, and Mohamed Younis. "Towards Autonomous Vehicular Clouds". In: May 2012. DOI: 10.1007/978-3-642-17994-5_1.

[5] "Federated Learning: Collaborative Machine Learning without Centralized Training Data". In: *Google AI Blog* (Apr. 2017).

[6] Ryan Florin et al. "Reasoning About Job Completion Time in Vehicular Clouds". In: *IEEE Transactions on Intelligent Transportation Systems* 18.7 (2017), pp. 1762–1771. DOI: 10.1109/TITS.2016.2620434.

[7] Takamasa Higuchi, Falko Dressler, and Onur Altintas. "How to Keep a Vehicular Micro Cloud Intact". In: June 2018, pp. 1–5. DOI: 10.1109/VTCSpring.2018.8417759.

[8] Seyyedali Hosseinalipour et al. *From Federated Learning to Fog Learning: Towards Large-Scale Distributed Machine Learning in Heterogeneous Wireless Networks*. June 2020.

[9] Yutao Huang et al. "When deep learning meets edge computing". In: Oct. 2017, pp. 1–2. DOI: 10.1109/ICNP.2017.8117585.

[10] Yann Lecun et al. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86 (Dec. 1998), pp. 2278 –2324. DOI: 10.1109/5.726791.

[11] Euisin Lee et al. "Vehicular Cloud Networking: Architecture and Design Principles". In: *IEEE Communications Magazine* 52 (Feb. 2014), pp. 148–155. DOI: 10.1109/MCOM.2014.6736756.

[12] Alberto Marchisio et al. "Deep Learning for Edge Computing: Current Trends, Cross-Layer Optimizations, and Open Research Challenges". In: July 2019, pp. 553–559. DOI: 10.1109/ISVLSI.2019.00105.

[13] Rui Pascoal, Ana Almeida, and Rute Sofia. "Mobile Pervasive Augmented Reality Systems -MPARS: The Role of User Preferences in the Perceived Quality of Experience in Outdoor Applications". In: *ACM Transactions on Internet Technology* 20 (Dec. 2019). DOI: 10.1145/3375458.

[14] Yaohua Sun et al. "Application of Machine Learning in Wireless Networks: Key Techniques and Open Issues". In: *IEEE Communications Surveys Tutorials* PP (June 2019), pp. 1–1. DOI: 10.1109/COMST.2019.2924243.

[15] Zeyi Tao and Qun Li. "eSGD: Communication Efficient Distributed Deep Learning on the Edge". In: *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*. Boston, MA: USENIX Association, July 2018. URL: https://www.usenix.org/conference/hotedge18/presentation/tao.

[16] Joost Verbraeken et al. "A Survey on Distributed Machine Learning". In: *ACM Computing Surveys* 53 (Mar. 2020), pp. 1–33. DOI: 10.1145/3377454.

[17] Xiaofei Wang et al. "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey". In: *IEEE Communications Surveys Tutorials* PP (Jan. 2020), pp. 1–1. DOI: 10.1109/COMST.2020.2970550.