Federated Learning for Autonomous Vehicles using FLWR

Sing-Yao Wu, Robin Song, Grant Liu

March 2024

1 Introduction

In our modern day, distributed systems are common place in many different and critical aspects of our life. Technologies, such as social media, cloud computing, banking, and in our case, autonomous vehicles, all rely on distributed systems to function properly. However, these systems bring a unique set of challenges due to their disjointed nature. Machines need to be able to communicate with each other reliably while maintaining atomicity. Latency, process failures, deadlocks, as well as a host of other problems all present their own challenges as well. With the rise of machine learning, it is understandable that these issues are further compounded by the desire to include neural network models in these systems. As such, specialized frameworks need to be developed to allow machine learning models to still function properly in a distributed system. Autonomous vehicles, especially, could not function without some sort of neural network model that can train the vehicles to understand the many complicated rules of the road. However, there are many implicit security risks with autonomous driving, and there needs to be a way to identify faulty information being fed into the model whether malicious or not. We also do not want data from individual vehicles shared with each other for privacy and security reasons. This is where we use the concept of federated learning. By using the FLWR framework, we can train a set of simulated autonomous vehicles to detect anomalies in car direction and velocity using simulated data from the Simulation of Urban MObility (SUMO). Utilizing this framework, individual nodes, representing vehicles, will train GRU (gated recurrent unit) models on the data set and send their parameters to a server, representing the roadside unit. The roadside unit will aggregate the parameters of the other models into a unified, central model, which we then test in terms of performance as well as the amount of latency experienced by the system.

2 Background

ROADSIDE UNITS Roadside Units (RSUs) play a crucial role in the development and training of autonomous vehicles by providing additional data and infrastructure support. They can collect data on traffic conditions, road surface conditions, weather information, etc. This data can be exchanged between RSUs and autonomous vehicles in real-time, providing valuable insights to enhance the vehicle's perception and decision-making capabilities. Critically, roadside units are necessary for federated learning frameworks in autonomous vehicles by serving as a server or connected to a server that can aggregate the data of multiple vehicles into a centralized model. Since vehicles should not have access to data from other vehicles, a central coordinator is necessary to maintain privacy and security. By leveraging the collective knowledge and data from RSUs deployed across an entire network, autonomous vehicles can benefit from more robust and generalizable models without directly accessing sensitive data.

FEDERATED LEARNING Federated learning takes multiple machine learning models over a distributed system and determines the ideal parameters for the model based on the values returned from each machine. The way these ideal parameters are determined can vary depending on the aggregation strategy deployed by the framework, but in our case we will be simply averaging the values. We will also be using a homogeneous framework where the same type of model will be used for every machine, making aggregation a simpler task. A more in depth explanation on federated learning can be read in section 2.1

2.1 Federated Learning

Federated learning [2] is a machine learning framework that allows for training models across decentralized devices that are holding local data samples without exchanging them. This decentralized approach offers several advantages, particularly in scenarios where data privacy and security are paramount. The federated learning process begins with the central server or aggregator initializing a global model, which serves as the starting point for training. This global model is then distributed to participating clients. Federated Learning uses a synchronous-round approach: In each round, the server chooses a fixed number of clients from clients that send requests to the server. The server then sends the model checkpoint to the selected clients and lets them train their models with the data on the devices. After training is finished, the clients will send their updated parameters back to the server so that the server can update the global model. This system is effective on models trained by data generated from the client side and data containing privacy information because the training data is never uploaded to the server. Also, there is usually a secure aggregation mechanism present, which helps prevent a server from guessing what kind of data a client has based on its model updates.

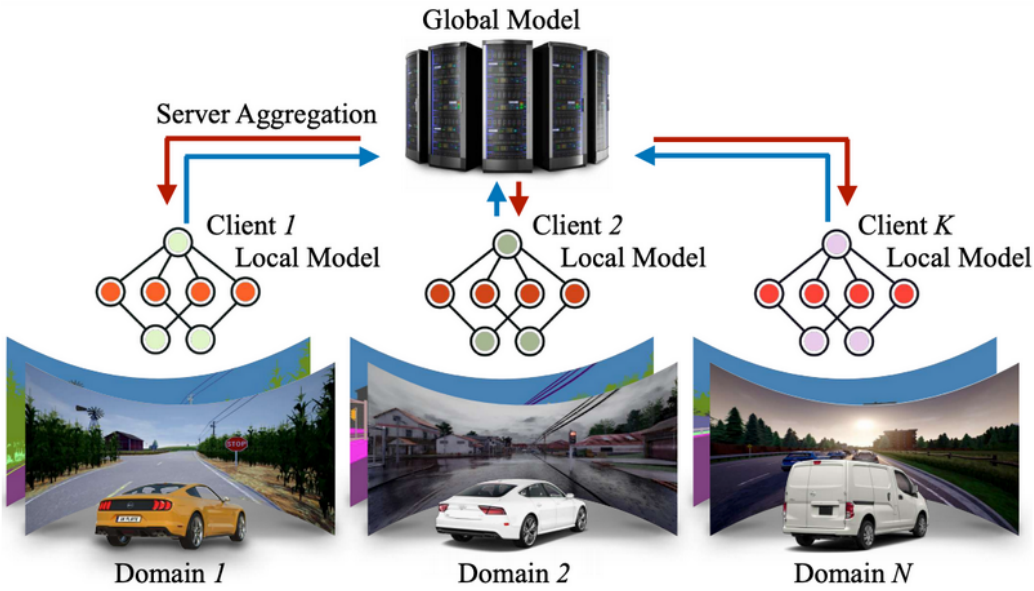

Figure 1: Federated Learning in Autonomous Vehicles: https://feddrive.github.io/

2.2 Security Against Bad Data and Bad Actors in Intelligent Vehicles

Data spoofing and manipulation are one of the potential security risks associated with autonomous vehicles. Bad actors can manipulate the information received by the vehicle's systems, leading to incorrect decisions or actions by the autonomous driving system. Federated learning can mitigate this risk by allowing vehicles to learn from decentralized data sources without the need to share raw data. Autonomous vehicles collect a vast amount of sensitive data about their surroundings, such as images, GPS coordinates, and other sensor data. Federated learning enables model training to occur locally on each vehicle's edge device without sharing raw data with a central server. This reduces the risk of data exposure and maintains the privacy of individuals whose data is being used. It also helps protect against data being intercepted, since raw data remains on the edge devices and is not transmitted to a central server. This distributed nature also helps mitigate the impacts of compromised units on the system as a whole. By training models using data from multiple sources, the impact of poisoned or malicious data is reduced. Even if a subset of devices is compromised by bad actors, the integrity of the global model can be preserved by aggregating contributions from a large number of devices.

3 System Design

3.1 Problem Formulation

In this project, we assume a predictor-based intrusion detection model [4] is used on all vehicles on a line. To improve the accuracy of trajectory prediction for higher intrusion detection accuracy, a federated learning process is conducted by a server to train the predictor model using the local trajectory information on each vehicle on the lane. For simplicity, we assume the server is just the roadside unit on the same lane while it can also be a server that communicates to multiple roadside units. The roadside unit selects a fixed number of client vehicles to perform local training in each training round, and aggregates the updated parameters received from client vehicles.

The client vehicles are equipped with sensors and V2V communications protocols to receive the trajectory information of itself and its preceding vehicle. The predictor model uses trajectory information to predict the trajectory in the future and compares it with the actual trajectory received later to detect the intrusion. Each vehicle stores the time series of the trajectory in its storage for future training. When training the local model, no additional manual labeling is needed because the labels of the input trajectory are also just the trajectories in the next few timesteps. Same as the settings in [4], in this project, we assume the trajectory information of each vehicle is its position, speed, acceleration, and the position, speed, acceleration of its preceding vehicle.

3.2 Implementation Detail

The implementation of our code can be found in `https://github.com/singyaowu/CS230Project_FL`.

3.2.1 Model Design

For simplicity, in this work, we use the same model structure as that in [4], which uses a two-layer GRU Gated Recurrent Units (GRUS) with a hidden size 120, connected with a Fully-connected layer and a *tanh()* function. The number of input trajectories is 40, and it predicts the trajectories of the next 10 timesteps. The main advantage of this model is that it has a small model size and relatively high speed, which is crucial for this project because most of the machines we have are not powerful enough to run complicated models. This is also reasonable because we usually hope the intrusion detection on vehicles can be run in real-time.

3.2.2 Federated Learning Framework

We use Flower [1] to build our federated learning framework. Flower is an open-source federated learning framework with high flexibility. It supports PyTorch and provides API for both remote procedure calls to different machines and simulations of a large number of clients on the same machine. The implementation details of our federated learning framework are as follows:

- The server uses federated averaging as the model aggregation strategy. This strategy averages the model weights received from all selected clients as the model weights of the server model.
- The federated learning training process is only activated on the server when equal to or more than two clients are connected to the server. This is to prevent from only one client being selected in the first round of training.
- The hyper-parameters of training on the client side such as learning rate or training epochs are assigned by the server.

4 Experiments

4.1 Experimental Setup

We use *Simulation of Urban MObility* (SUMO) [3] to simulate the vehicle flow on a lane. We created three types of vehicles with different characteristics such as different max speed, acceleration/deceleration, car length, car-following models, and the possibility of occurrence, and built three scenarios based on the lane and three vehicles, as shown in Figure 2. In each scenario, only one type of vehicle runs on the lane. We simulate each scenario for 100000 simulation steps and get three trajectory datasets with different

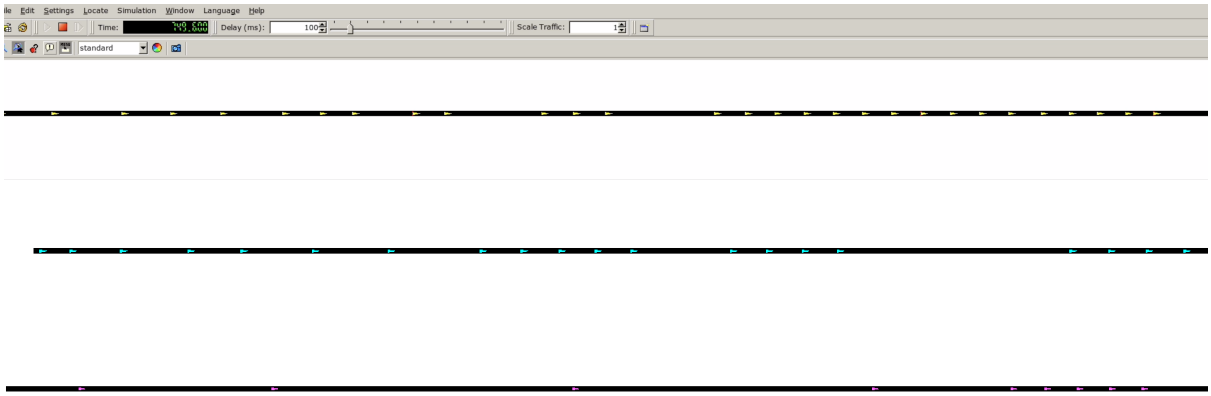

Figure 2: Three types of vehicles, each with different car-following models (CACC, ACC, Krauss), densities of car flow, and other configuration settings.

vehicle properties. These datasets are then split into the training dataset and the testing dataset with a ratio of 4 to 1.

The training datasets assigned to each client are done in two ways: uniform data distribution or heterogeneous data distribution. Assume the number of the clients is n. In the uniform data distribution, the dataset of each scenario is split equally into n partitions with equal size and assigned to each client, so all clients get the same size of data in each scenario. In the heterogeneous data distribution, the data of three scenarios are first concatenated together in order and are then split into n partitions and assigned to each client. Therefore, each client can only get the data from one or two scenarios.

We perform two types of experiments: federated learning simulation on a single machine, and federated learning on multiple machines. For simulation on a single machine, we perform it on a desktop with 11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz. We simulate 100 clients connected to 1 server and run for 100 rounds. In each round, the server selects 10 clients and sends the weight of the current model and the hyper-parameters to these clients. Each client trains its dataset with 1 epoch, and a batch size of 20. The simulation is tested on both the uniform data distribution and heterogeneous data For federated learning on multiple machines, we used the desktop mentioned previously and a MacBook Pro with Apple M1 Pro 10-core and 16-core GPU as the clients, and another desktop with Intel(R) Core(TM) i9-13900KF @ 5.8GHz and NVIDIA GeForce RTX 4070 Ti as a server, and run for 20 rounds to measure the accumulated runtime and loss change on uniform training data distribution.

4.2 Centralized Learning

We first trained our models using our simulated data in a centralized setting for comparison, where one machine handles all the training. In terms of the data that are used during training, we are considering 4 different scenarios: the first three use each type of data individually, and the last scenario is the combined dataset. We employed the Adam optimizer with a learning rate of 0.0001 and a Mean Squared Error (MSE) loss function. The model was trained for 40 epochs with a batch size of 128. Figure 2 shows the training loss and validation loss during each epoch.

4.3 Federated Learning on Multiple Machines

A screenshot of the demo on multiple machines is shown in Figure 4, and the result is shown in Figure 5. The left-hand side of Figure 5 shows the accumulated runtime in each round of server. The runtime is recorded on the server side when receiving responses from all the clients. The accumulated runtime can indicate the potential latency between communication and the overhead of training on the clients' side. The trend of the accumulated runtime is linear, which is reasonable because there are only two clients in the system, the server always can only select the same clients. The result can change if we have more machines for the server to select from. Also, because of the design of the federated learning network, the runtime of each round is bottlenecked by the slowest client machine. We've tried to add a third laptop with lower computational power, but found that the runtime of each round still increases dramatically even though the training data assigned to each client is less than that in the two-machine

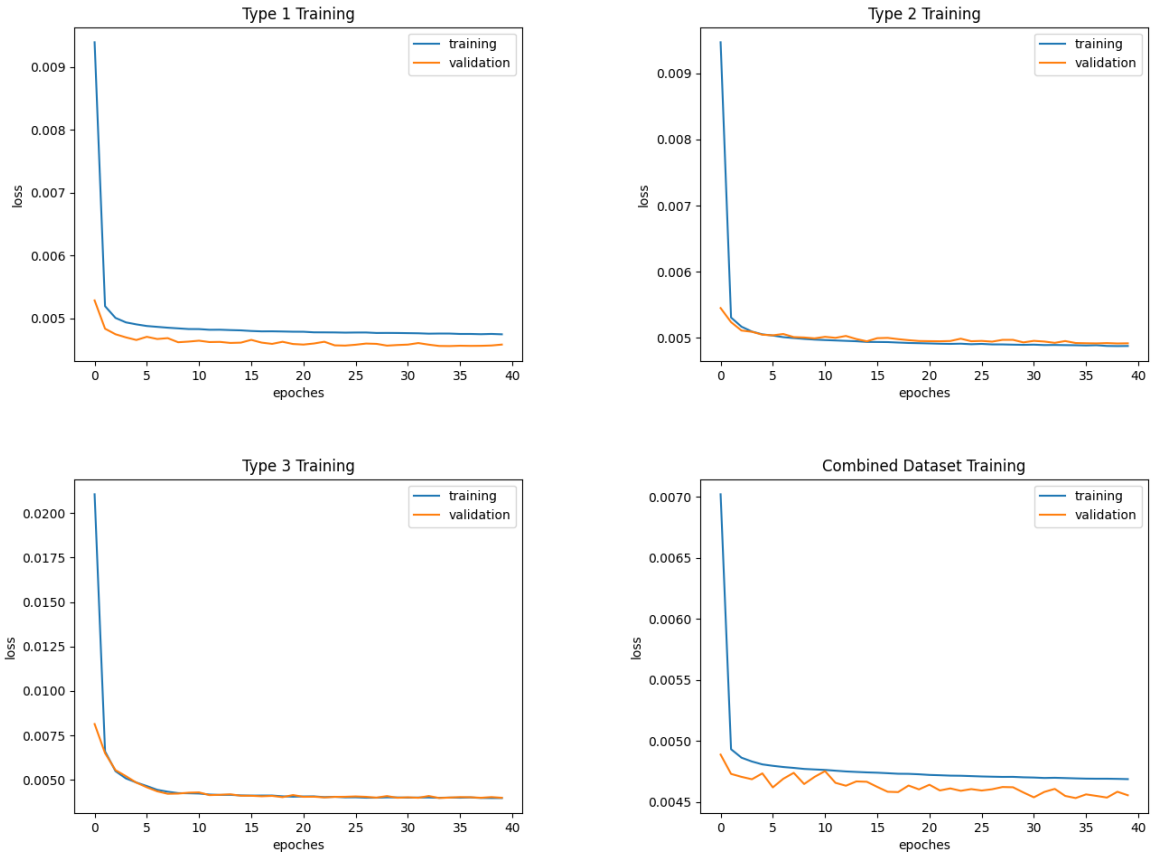

Figure 3: Losses During Centralized Training

settings. Therefore, in the real-world use case, the design of the timeout of waiting on a client's response can be crucial to the overall performance of training.

The right-hand side of Figure 5 shows the mean loss of the server and two clients in each round. The loss of the server is similar to the loss in the centralized training in Figure 3 in the same number of rounds and epochs. While the loss of the server model is not the lowest in the first three rounds, it becomes the lowest after the fourth round, which satisfies our objective to aggregate weights of client models for better performance. An interesting finding is that while the difference between the loss on both clients converges to similar losses when the number of rounds increases, the difference between that on the client models and on the server seems to increase as the number of rounds increases. We think this is because the client model is always updated to the server model every round, which minimizes the loss of the whole dataset. However, the loss of the client model is measured through the validation dataset, which has less data than the testing dataset on the server model. Therefore, the difference in loss may indicate the bias between the validation dataset and the testing dataset.

4.4 Federated Learning Simulation on Single Machine

A screenshot of the demo on simulation is shown in Figure 6, and the result is shown in Figure 7. The left-hand side of Figure 7 shows the runtime of simulations with 100 clients in the uniform training dataset scenario and the heterogeneous dataset scenario. The runtime difference in the two conditions is small, and the heterogeneous setting seems to have slightly more latency. Based on our knowledge, the data distribution shouldn't affect the throughput, so this is probably because of some difference in the background process of the machine. Compared to Figure 5, the throughput is higher by about 5x. This is mainly because in the single-machine simulation, while the training dataset is equally allocated to all the 100 clients, only 10 clients are run in one round. Compared to the multiple machine settings where each client machine trains with half of the training dataset, the simulation machine only trains on 1/10 of it, creating an illusion speedup of 5x.

The right-hand side of Figure 7 shows the mean loss of simulations with 100 clients in the uniform

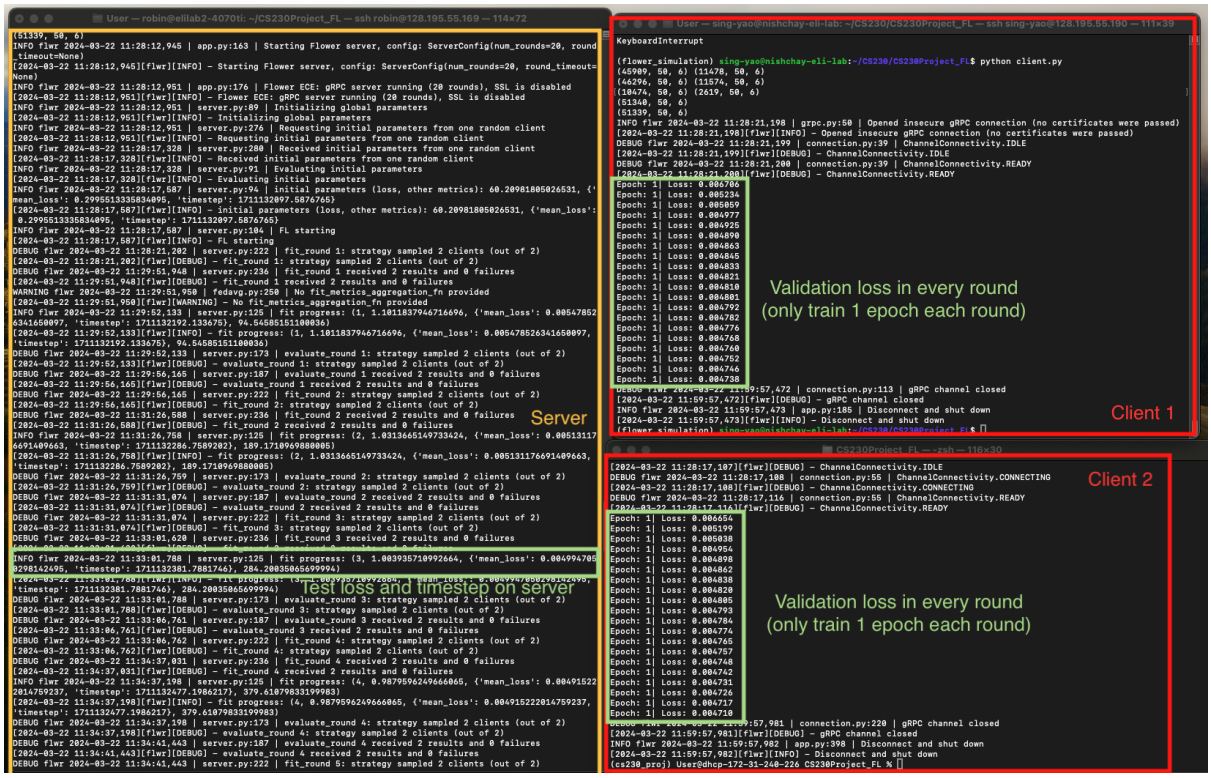

Figure 4: Screenshot of the Execution on Multiple Machines. The validation and test loss are shown in each terminal.

training dataset scenario and the heterogeneous dataset scenario. Unlike the loss curve in the uniform setting, the loss curve in the heterogeneous setting fluctuates frequently. The reason is that the selected clients in every round are different, so the data distribution of the three scenarios in all the training data of the selected clients varies in every round, resulting in unstable training behavior.

5 Discussion

The results from our Centralized Learning and Federated Learning are comparable in terms of loss, and the Federated Learning allows the data to be inside each client without sending it to the client. This preserves data privacy in a decentralized setting without loss of accuracy; however, there are still some challenges and issues.

Unbalanced performance of machines. Since the server needs to wait for all the distributed data to be trained before starting the next round, the training latency is bottleneck-ed by the slowest machines. When the fast machines finish their training, it can only wait for the slowest one to finish before getting the next round of model parameters from the server. One possible solution can be adding timeouts for each training round. However, the timeout adjustment can be difficult because it is hard to predict the average computational power of the clients in each round, and thus, the timeout can be dynamic.

Heterogeneity of data in each client. In the real world, each vehicle can be different in many aspects (i.e, maximum speed, acceleration). These variations can introduce unexpected challenges in aggregating model updates, and it is impossible to use one model to fit the data in all the clients. Therefore, customized models for different vehicles may be required and this further complicates the federated learning structure.

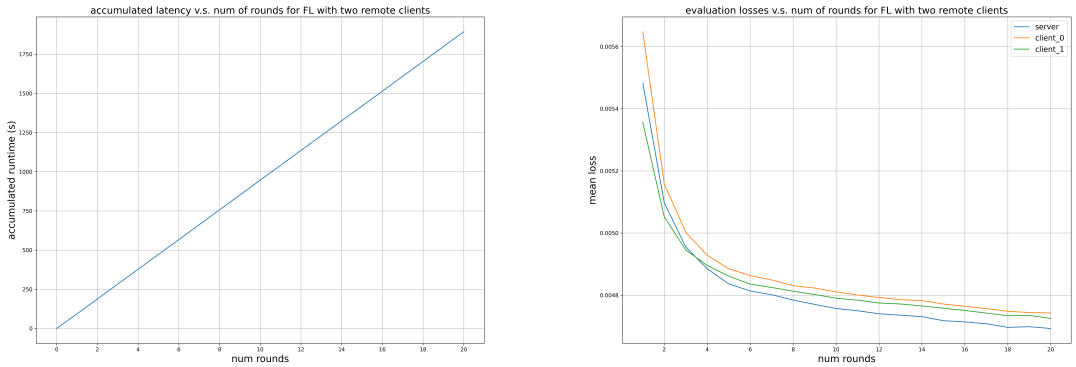

Figure 5: Accumulated Runtime and Mean Loss during Federated Learning on Multiple Machines

6 Conclusion

In this project, we introduced a federated learning application for intelligent vehicles. We integrated the predictor-based intrusion detection model for the vehicle-following system into the federated learning framework and experimented with different scenarios. The experimental result shows the possibility of implementing the applications in real-world settings. It also shows the problems that may occur when performing federated learning such as throughput bottleneck on the slowest machine and loss fluctuation on heterogeneous training data distribution. For future work, we may consider more security issues on federated learning, such as secure aggregation and secure multi-party protocols.

References

[1] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchi Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2022.

[2] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloe Kiddon, Jakub Konečnỳ, Stefano Mazzocchi, Brendan McMahan, et al. Towards federated learning at scale: System design. *Proceedings of machine learning and systems*, 1:374–388, 2019.

[3] Pablo Alvarez Lopez, Michael Behrisch, Laura Bieker-Walz, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Leonhard Lücken, Johannes Rummel, Peter Wagner, and Evamarie Wießner. Microscopic traffic simulation using sumo. In *The 21st IEEE International Conference on Intelligent Transportation Systems*. IEEE, 2018.

[4] Sheng-Li Wang, Sing-Yao Wu, Ching-Chu Lin, Srivalli Boddupalli, Po-Jui Chang, Chung-Wei Lin, Chi-Sheng Shih, and Sandip Ray. Deep-learning-based intrusion detection for autonomous vehicle-following systems. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 865–872, 2021.

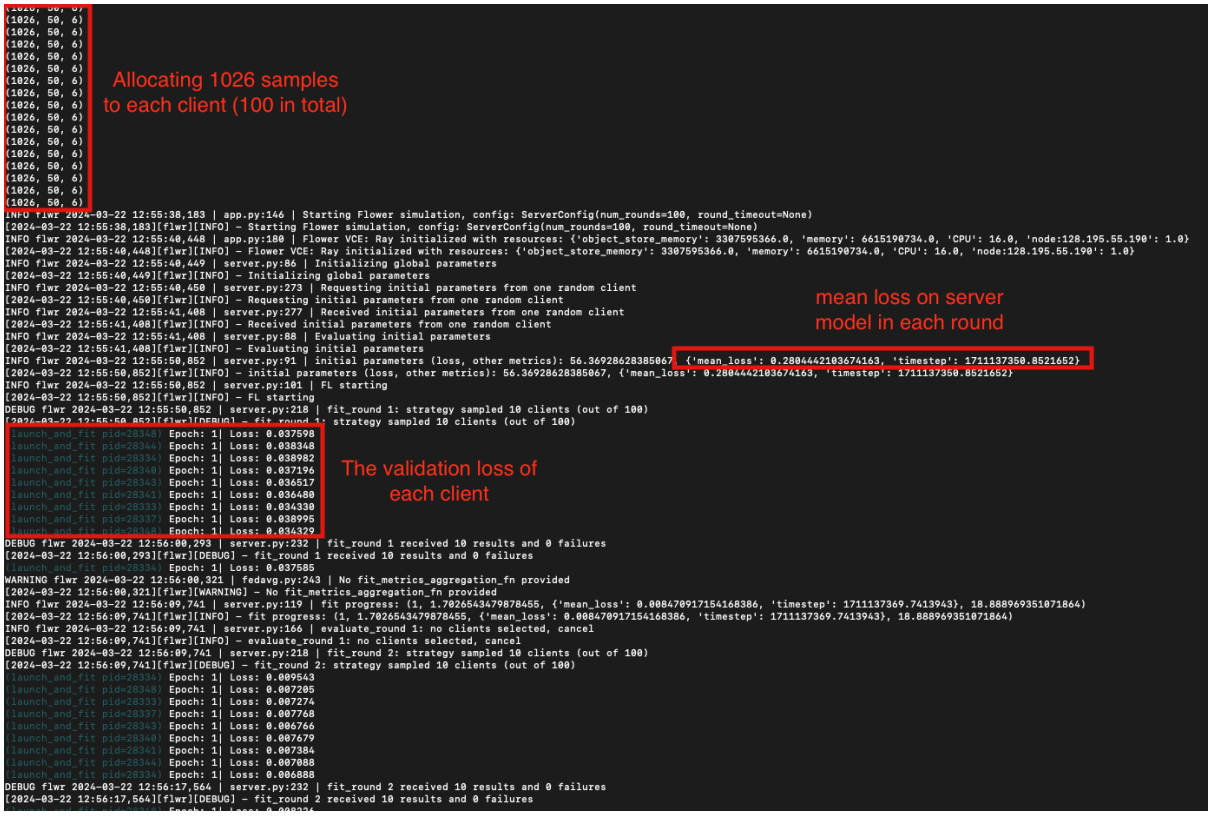

Figure 6: Screenshot of the Execution on simulation. The validation and test loss are shown in each terminal.

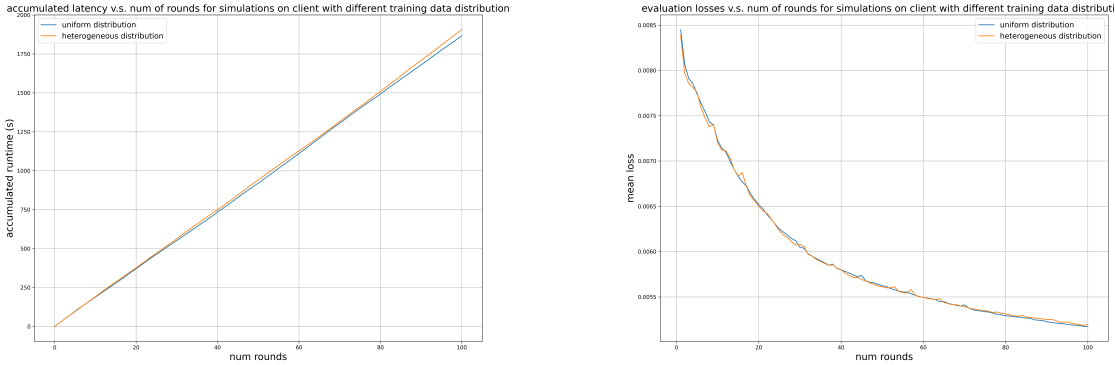

Figure 7: Losses and Latency During Federated Learning on Simulated Machines