Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Towards production-ready end-to-end Federated Learning for automotive applications

Swetha Lakshmana Murthy

Course of Study: Information Technology

Examiner: Prof. Dr. Marco Aiello

Supervisor: Dr. Denis Kramer, Bosch.IO GmbH

Commenced: May 16, 2022

Completed: November 15, 2022

Acknowledgement

I'm ecstatic at witnessing my Master's thesis come to fruition, and I want to express my gratitude to everyone who has been instrumental throughout my journey.

I would like to express my gratitude to Bosch.IO GmbH and the Department of Service Computing at the University of Stuttgart for allowing me to pursue this thesis.

I would like to express my sincere and immense gratitude to Dr. Denis Kramer for the incredible support and encouragement in every walk of the thesis. He has been pivotal in molding my learning curve throughout the thesis work with his insightful advice, feedback, motivation, and unwavering support.

I would like to convey my deepest gratitude to Prof. Dr. Marco Aiello for his encouragement, amicable support, providing feedback, and examining my thesis work.

I would like to thank my parents, sister, grandfather, and friends, who have always been my Rock of Gibraltar. It would have been impossible to complete my thesis without their love and support.

Abstract

The growing trend toward preserving user data privacy has embraced a new computing paradigm called Federated Learning (FL). FL enables edge devices to learn from a global shared model without cloud data storage. FL implements an efficient model training method for disseminating model training while maintaining user data privacy. Applying FL to the automobile industry solves numerous problems, including minimizing privacy concerns, establishing clients' trust in data sharing, and constructing robust ML models. The time it takes to collect data is also mitigated because we do not have to wait for clients to share their data. Instead, the training at the edge node is enough.

Meanwhile, the automotive industry is experiencing a dramatic increase in the popularity of Electric Vehicle (EV). In providing greener technologies, EVs offer substantial advantages. The batteries that power the EVs have a maximum energy consumption restriction. Consequently, it is vital to evaluate the battery consumption of EVs to resolve numerous issues, such as range anxiety and Charging Station (CS) usage, to name a few. Therefore, it is crucial to predict battery consumption under various traffic scenarios. Another critical issue to be tackled is the electric load forecasting of CSs in an area. This is especially important due to the frequency with which EVs utilize CSs.

To end this, we implement an amalgamation of the FL in the automotive use cases. We implement an FL environment to embark on the battery capacity estimation of EVs and the energy demand of CSs in a specific region. We used the Simulation of Urban MObility (SUMO) simulator's various toolkits to model the traffic, network simulation, and data acquisition. We use Flower, an open-source FL framework, to set the ground for our method. We implement and adapt the software stack in Flower to accommodate the two user-defined strategies, active client contribution (client picking strategy) and handling faulty clients (system fault tolerance). We eventually integrate the established FL into an experiment tracking tool called Weights and Biases (W&B). The implemented FL system runs a large client pool of about 1000 EVs, enabling a large-scale production-grade and scalable simulation in FL. The simulation environment is configured as resource-aware using the Virtual Client Engine (VCE) of Flower and Ray. Our experiments show significant convergence results preserving data privacy. The findings show that the FL model outperforms the local baseline model in many client cases. We also demonstrate the implementation of a fault-tolerant system using an extension to the Federated Averaging (FedAvg). Finally, we present the significance of adopting a differentially private model and the trade-off between end-to-end privacy protection and obtained accuracy.

Contents

List of Figures

List of Tables

List of Listings

Acronyms

BEV Battery Electric Vehicle. 27

CAVs Connected and Autonomous Vehicles. 34

CS Charging Station. 5

DL Deep Learning. 15

DP Differential Privacy. 23

EV Electric Vehicle. 5

FedAvg Federated Averaging. 5

FL Federated Learning. 5

FVN Federated Vehicular Network. 27

IID Independent and Identically Distributed. 23

ITS Intelligent Transportation Systems. 34

LSTM Long short-term memory. 25

MAE Mean Absolute Error. 54

ML Machine Learning. 15

MLOps Machine Learning Operations. 16

MSE Mean Squared Error. 54

RMSE Root Mean Squared Error. 55

RNN Recurrent Neural Networks. 25

SGD Stochastic Gradient Descent. 22

SOC State of Charge. 16

SUMO Simulation of Urban MObility. 5

TFF TensorFlow Federated. 29

V2X Vehicle to Everything. 34

VCE Virtual Client Engine. 5

W&B Weights and Biases. 5

1 Introduction

Modern edge devices like mobile phones, wearable devices, and autonomous vehicles use cutting-edge technologies to power a variety of functional applications [48]. The enormous amount of data generated by these edge devices is crucial for today's rapidly developing technological advancements. With the rising computational power of these devices and the security concerns associated with transmitting private information [30], storing data locally and shifting network computation to edge devices is becoming an increasingly attractive option. The requirement for such data to build a reliable, comprehensive Machine Learning (ML) and Deep Learning (DL) platform is expanding tremendously. The idea of user privacy does, however, frequently place restrictions on using them. We have crucial information in fields like the automotive industry, healthcare units, and educational platforms, but the user is wary of sharing it with the outside world. Employing centralized training would seem difficult because we cannot send the data to the central storage unit for training owing to privacy issues.

To this end, we introduce the concept of the developing computing paradigm known as Federated Learning FL [36, 32, 44, 50]. FL aims at preventing data sharing with the other participants or the central storage unit, keeping it private to each user [44, 32, 50]. On a brief outline of the FL setup, the client that is a participant shares the individual updates and receives a global model update from the central server based on final aggregation from all the other clients. Each client uses these updates for the next round of federated training. A prevalent instance of an FL application is the Gboard [24], a Google keyboard where FL is used to deal with sensitive user data. Another example is the medical sector, where obtaining consent to upload data to the cloud may be time-consuming for ethical and regulatory reasons [73]. In these instances, FL can be used automatically to address the issue of data availability and assist in developing a more effective privacy-protected ML system. FL finds its usage in multiple domains [41]. The following sections discuss the extensive use of FL in the automotive industry, which is the topic of interest here.

1.1 Motivation

Emerging vehicular IoT systems comprise significant sensor data. With numerous connected vehicles, edge computing in vehicular networks is rapidly evolving. Furthermore, vehicular networks are highly dynamic. Their environments change over time, and multiple scenarios contain private information. Thus, employing FL in such scenarios is paramount to the data acquisition infrastructure that has distributed nature and prevents privacy issues [95]. To fully mold the potential of a more ingenious vehicular IoT system, evaluating AI methodologies is required to meet growing demands and build efficient machine-learning models.

Researchers have started leveraging the concept of FL in the vehicular automotive industry to tackle the above-stated concerns. In the recent past, automotive FL has made tremendous global contributions [61]. In FL automotive systems, no data is transmitted outside the vehicle, as all

learning is performed locally on the edge device. The shared weights do not reveal the user's identity during the FL process. Developing FL for automotive units necessitates a production-grade environment for continuous monitoring and distributed deployment of ML models. In light of these factors, it is clear that FL has the potential to become significant in the automotive industry. Figure 1.1 illustrates a general framework of FL in vehicular edge computing where the local model updates are sent from the respective vehicular clients, and other clients can download the global model.

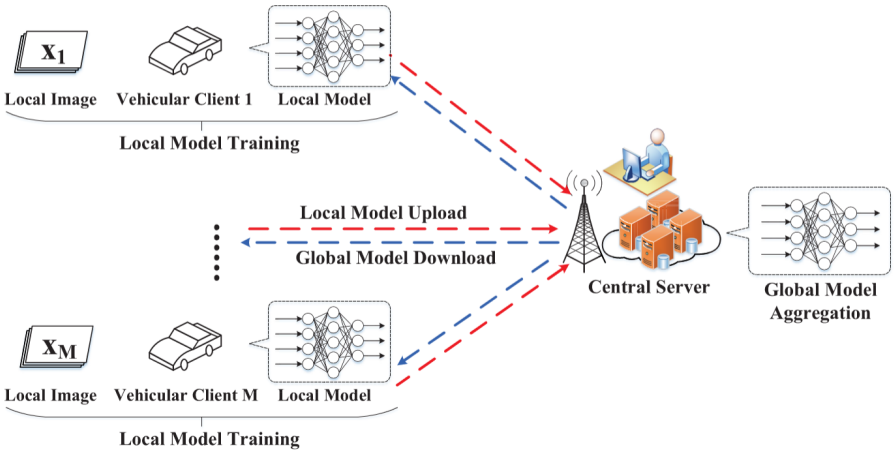

Figure 1.1: A general framework of FL in vehicular edge computing [88]

1.2 Problem Statement and Research Questions

According to the study [83], we see an exponential increase in Electric Vehicles (EV) usage in Germany. EV strive to alleviate massive energy consumption and drastic climatic shifts [49]. However, the growing usage of EVs has also led to customer data sharing, and privacy issues [78]. Central storage of all such EV data is a conundrum. The EV industry is increasingly concerned with addressing user data vulnerability and developing a complete setup ready for production. On the other hand, the power grid provides energy for Charging Station (CS) once EVs have initiated charging. Conventionally, the CS' energy consumption data is again stored centrally. This approach again lacks privacy leading to issues in privacy for both the CSs and EVs. Also, EVs predominantly use battery stacks based on lithium-ion (Li-ion) cells. It is essential to measure the State of Charge (SOC), defined as the available capacity and expressed as a percentage of its rated capacity, to assess the potential energy of a battery [58].

A fully functional FL requires selecting a framework that supports a comprehensive software stack, end-to-end privacy guarantee, and integration with the Machine Learning Operations (MLOps) platform. Several FL frameworks exist; however, selecting a framework that offers scalability, algorithmic development support, and system-level research approaches is essential. Understanding and implementing these framework environments in a production-grade environment is critical. The wherewithal to implement such client-friendly, production-scale FL will unleash new avenues for study in the automotive sector.

In this work, we mainly address the above-listed occurring privacy issues in the production-grade automotive industry by formulating, discussing, and answering the below research questions (Table 1.1),

Table 1.1: Research Questions

	Research Questions	Motivation behind RQs
RQ1	How can we incorporate the FL environment justifying user privacy and system heterogeneity to realize use-cases (a) and (b)? (a) Actual battery capacity prediction for EVs (b) Energy-demand forecasting for CSs	Using FL in the automotive industry to preserve client data privacy
RQ2	How does the above proposed system system provide incentives for deploying FL in a production-ready automotive environment?	
	Sub-questions	
	RQ2.1 *How does the system handle the training and evaluation of numerous clients?*	Large-scale Simulation
	RQ2.2 *How do we implement a user-friendly client-picking strategy for active client contribution?* ***RQ2.3*** *How can we handle faulty clients during the training and evaluation process?*	Customized Strategies
	RQ2.4 *How can we integrate the proposed FL system with a production-grade MLOps platform for experiment tracking?*	Experiment Tracking
RQ3	How can we additionally enhance the privacy guarantees for the local client model during weight updates using differential privacy?	End-to-end privacy guarantee

1.3 Contributions

This thesis aims to develop two prevalent use case scenarios for the EV and CS automotive industries. The method presented here aims to simulate a federated environment for EVs and CSs in a given geographical area to compute the individual battery capacity of EVs and the total energy demand from various CSs in the given zone. We intend to create a large-scale, production-ready FL simulation environment that addresses multiple elements, including client selection, fault tolerance, results aggregation, and privacy guarantees. We integrate the FL environment further into a developer-friendly MLOps platform for tracking and benchmarking the metrics.

Due to the impracticality of collecting such a vast amount of EV and CS data for early development, we use simulators to create a realistic design. We also investigate the various semantics of the employed simulator.

Precisely the research study presented in this work addresses the following contributions,

- Design, traffic, and mobility model, to set up a test bench for EVs and CSs and obtain the simulation data.
- Leverage a federated framework and implement use cases with privacy protection for EVs and CSs.
- Illustrate the nature of data and the large-scale client pool simulation using the FL framework.
- Develop the implemented system further to include user-tailored strategies, the algorithm to handle faulty clients, resource allocation for single-node simulation for heterogeneous large-client cohorts, and end-to-end privacy guarantee for local models.
- Track the various metrics using an experiment-tracking MLOps tool.

1.4 Preview of Chapters

From preliminary concept to result analysis and discussions, the contribution of this thesis is highlighted below,

- *Chapter 2 - (Background)*
  This chapter explains the relevant concepts and their prominence in FL. We discuss and provide an overview of the other common topics related to the thesis.
- *Chapter 3 - (State of the Art)*
  This chapter provides an overview of the various FL-related automotive developments that have been handled so far. We also discuss the state-of-the-art in this area and why FL is necessary for the automotive industry. We also discuss the merits and demerits of different FL frameworks, followed by the core design architecture description of the chosen framework. The following section outlines the related works for the traffic and mobility simulator. On a concluding note, we discuss the derived insights and contributions.
- *Chapter 4 - (Methodology)*
  This chapter outlines the design methodology for the work proposed. It introduces the two use-case scenarios that we implement here. We discuss the different semantics of the simulation tool and methods to acquire the EV and CS data. Following this, we describe the FL settings in detail. The sections in this chapter collectively introduce the idea conceptually and mathematically.
- *Chapter* 5 - (*Experimental Set-up*)
  This chapter discusses the realization of our FL system. We discuss in depth the experimental setup for data acquisition and the implementation of the FL environment leveraging the design concepts introduced in the previous chapter.
- *Chapter* 6 - (*Results and Observations*)
  Using the obtained results, we also provide a rationale for the research questions based on the outcomes.
- *Chapter* 7 - (*Conclusion and Outlook*)
  This chapter lists the concluding remarks, limitations of the work, and the research questions open for further analysis.

2 Background

2.1 Federated Learning

Traditional centralized distributed learning has limitations, such as low efficiency with high transmission costs and lack of privacy protection, which reduce ML and DL application levels in various industries and medical institutions, among others [102]. In addition, a large amount of training data, processing capacity, power consumption, and enough resources for computing are required to ensure many sectors implement machine learning. Moreover, most organizations are not interested in sharing their data due to security and privacy concerns.

With the development of digital technology and a heightened awareness of the need to take safety precautions, consumers' anxiety regarding preserving their privacy is growing, creating a unique concern regarding the data island problem [23] in IIoT networks. Unfortunately, a significant amount of data is necessary to take advantage of the capabilities for examination and understanding offered by ML and DL technologies. Therefore it is vital to solving the data island problem.

In order to solve collaborative learning circumstances within enterprises, the concept of decentralized collaborative machine-learning algorithms that preserves anonymity was introduced. Federated Learning (FL), also known as federated machine learning, is one framework for ML that utilizes data and executes ML without sharing local data. FL encompasses data partitioning, security, and applications within itself. The workflow and architecture of the FL system are as in Figure 2.1.

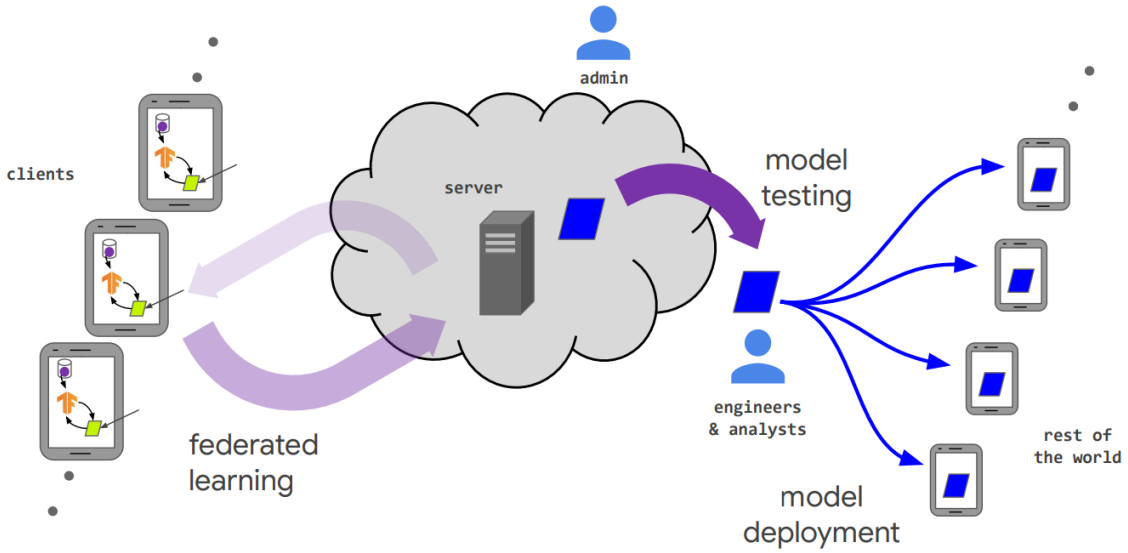

Figure 2.1: The lifecycle of an FL-trained model and the various actors in a federated learning system [32, 5]

2.1.1 Overview of FL

The definition of FL can be described as follows [32],

*"**Federated learning** is a machine learning setting where multiple entities (clients) collaborate in solving a machine learning problem, under the coordination of a central server or service provider. Each client's raw data is stored locally and not exchanged or transferred; instead, focused updates intended for immediate aggregation are used to achieve the learning objective."*

Mathematical definition of FL
Following is a mathematical description of the definition of FL [5]. All client data is aggregated and stored on a central data server in a centralized ML setting.

The clients or data owners can be represented as, $\{F_1, F_2, F_3, ..., F_N\}$ where N is the number of clients participating in the FL process. Their respective data is represented as $\{D_1, D_2, D_3, ..., D_N\}$. Assume that each client sends data D_i to the cloud. Consequently, these aggregated data D = $\{D_1, D_2, D_3, ..., D_N\}$ are utilized for training the model $M_{centralized}$ in a centralized ML setting.

In contrast, local updates from all data owners are used in an FL environment to train a global model termed $M_{federated}$. It should be highlighted that no data owner F_i is obligated to disclose their data D_i with others.

FL offers several capabilities concerning privacy and other problems encountered in computing paradigms. As we do not transfer the data via the network, the possibility of unauthorized data access is eliminated. There is mitigation of network traffic because training outcomes are typically far less substantial than the data. The time and cost of information transport can be reduced by lowering the quantity of transferred data. The requirements for the central processing cluster and central storage are less, as storing all data in a single location is optional.

Generic steps involved in FL training

An FL system consists of three key components: a central server where model updates are received and aggregated into a global model, clients that retain their data, and a computing framework that establishes the flow of client-server communication.

The FL process typically comprises the following steps [34, 14],

- Initial Stage -
  At first, understand the need for FL for the scenario defined and comprehend the overview of the nature of data distribution across the clients.
- Setting-up FL framework -
  Understand and select/implement a necessary framework for FL implementation.
  Understand the core architecture of the framework (Strategies, algorithm, hyper-parameters, evaluation metrics, etc.)
- Client selection -
  The central server selects the clients based on the implemented strategies. Other aspects include data distribution, client node characteristics, model training requirements, and training methodologies during client selection.

- Model dissemination -
  Configure the client-server communication.
  The initial model updates are sent to the chosen clients from the central server.
- Distributed learning using client feedback -
  After the first round of training, send local model updates to the central server.
- Aggregation -
  The central server computes a global model by aggregating the updates from the client nodes using an FL algorithm.
- Evaluation Phase -
  Download the aggregated global model.
  Evaluation of the clients chosen in the FL system.
- Continuation of the FL process -
  Continue the above steps until we meet the necessary convergence rate.

2.1.2 FL Applications

As was previously said, FL is a powerful distributed ML approach that preserves user privacy. It tackles numerous challenges, including data ownership, privacy, security, and different data dissemination, and thus has applications in numerous areas [41]. FL has many supported use cases, including the autonomous sector, healthcare, wearable, industrial IoT and management, and the financial sector. Following are a few prominent application examples for FL.

Google Keyboard (Gboard): One of Google's earliest innovative deployments of the FL paradigm, the Google keyboard (Gboard) [24], anticipates the next word on the smartphone's virtual keyboard, thereby protecting user data. They adopted the FedAvg algorithm, with SGD used for server-client training. This initial investigation demonstrates the practicality of training models locally on end-user devices instead of transferring sensitive user data to servers granting complete user control over their privacy protection. Integrating privacy by default with distributed training and aggregation across several client devices makes it more unrestricted.

The FL flow in smartphones' next-word prediction task is illustrated in Figure 2.2. Rather than transmitting the raw data to the central server, a distributed technique is utilized here. In this configuration, remote smartphones frequently communicate with a central server to send and receive their respective updates. During each round of communication, a subset of selected phones does local training on their non-identically dispersed user data and communicates these local updates to the server. After incorporating the updates, the server retransmits the new global model to a subset of smartphones. This network-wide iterative training procedure continues until convergence is achieved or a stopping requirement is met.

Medical Applications: FL has become the standard decentralized learning framework for medical applications in contexts that ensure privacy when training data is not immediately available. As a direct consequence, FL has demonstrated significant potential in the medical field [68, 70]. It has proven to be extremely helpful in brain tumor segmentation [82], cancer diagnosis [11] such as breast cancer detection [31, 73], fMRI screening and analysis [47], and keeping patient medical records [28], to name a few of these possible applications.

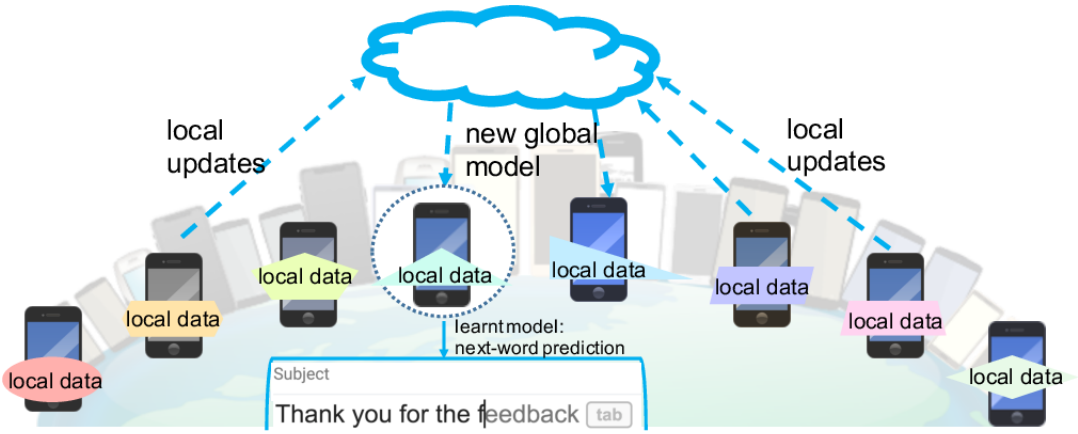

Figure 2.2: Example illustration of using FL for next-word prediction in smartphones [44]

Finance Sector: The banking and auditing industries have undergone dramatic shifts due to digital developments. The availability of vast amounts of data from various clients gives AI a disproportionately large role in developing this industry. However, substantial data confidentiality procedures must be observed when gathering data from clients and organizations. Results show that large-scale DL models are susceptible to leaking such private data. In such scenarios, we can employ the benefits of FL. One such work here protects the confidentiality of individuals in audit companies [81]. The proposed model learns client-beneficial representations without compromising confidential data.

2.1.3 FL Algorithms

FL utilizes multiple strategies to aggregate the global model [42]. McMahan et al.'s Federated Averaging (FedAvg) [55] is the most often used algorithm in the FL process. It is employed widely due to its ease of use and inexpensive communication costs. A random selection of clients, the aggregation of local model updates, and the formation of a global model constitute the general operation of FedAvg. It combines Stochastic Gradient Descent (SGD) performed on the selected clients and thereupon aggregates and averages the sequences in the server to form a global model after every iteration round [46]. Several modifications to the FedAvg algorithms provide numerous alternative algorithms, including FedProx, QFedAvg, FaultTolerantFedAvg, and FedNova, among others, to improve FL model performance. The advancements offered by FedAvg are beneficial to a number of server-side optimization algorithms, including FedAdagrad, FedYogi, and FedAdam [69].

To address system heterogeneity, FedProx [45] was proposed as a generalization and re-parameterization of FedAvg. It is another widely employed algorithm. It consists of an additional proximal term that enhances system stability by reducing the deviation of the averaged model from global optima.

Using the SCAFFOLD algorithm [33] that employs the variance reduction technique, the drift in each client's updates, which causes slow and unstable convergence, can be corrected.

FedNova [93] is a method of normalized averaging that eliminates objective inconsistency while maintaining rapid error convergence. FedProx and SCAFFOLD improve the training phase of FedAvg, whereas FedNova improves the aggregation phase. Before averaging, it considers the heterogeneous local updates of each client and normalizes the local models accordingly. QFedAvg [43] promotes fairness by encouraging fairer accuracy distributions across devices in FL.

2.1.4 FL Concepts and Challenges

We highlight the three main concepts in FL and their pertaining challenges [5] that are widely seen throughout the research work,

Statistical Heterogeneity: Devices frequently generate and collect data that is not uniformly distributed throughout the network. The number of data points collected by devices can vary substantially. Furthermore, the underlying data structure for each client may be very different from one another. In addition, the feature space may change, resulting in a different set of data for different users. This data type contradicts the existing concept of Independent and Identically Distributed (IID) assumptions, resulting in non-IID data [103, 12]. This lack of IID increases the likelihood of system laggards. During model aggregation, severe weight divergence caused by device heterogeneity data can also be detrimental [96].

Several classifications, including attribute, label, temporal, and quantity skewness [103], are used to categorize non-IID data. Numerous studies have been performed to improve the stability of models for such data [101, 45]. In the following chapters, we discuss these categories of non-IID mathematically for the data employed in this study.

System heterogeneity: In federated networks, each device's storage, computation, and communication capacities may vary due to variances in hardware, such as CPU, RAM availability, network access, and power. Each device may be unstable, and it is prevalent for an active device to cease functioning at a particular iteration due to connectivity or energy constraints. In addition, the network size and system-related limitations on each device frequently result in just a tiny proportion of devices being active concurrently, for instance, hundreds of active devices in a network of one million. It is necessary to configure clients in FL effectively.

It is crucial to modify the system architecture for asynchronous learning, enabling users to contribute their device resources and local data to train a shared machine learning (ML) model [56]. The added benefit of joining and leaving the system at any moment without interrupting the learning process should be considered while adapting to such a fault-tolerant system. The following chapters will discuss the management of system resources and fault tolerance.

Privacy-preserving mechanisms: Even though FL handles privacy by storing data on each client's device, the privacy issue remains pervasive. Numerous attacks have been against machine learning models that might deduce the raw data by accessing the model. Instead of storing or transmitting the raw data to the server, FL sends the trained weights or gradient computations. Although local data are not accessible in FL, the transferred model parameters may disclose critical information. In scenarios, various privacy techniques, such as Differential Privacy (DP) [15] and k-anonymity [87], to name a few, offer varying privacy guarantees. We discuss one such privacy-preserving technique, DP using the Opacus [98] library later in the upcoming sections.

2.1.5 Differential Privacy

After each round of training, preserving the privacy of the local client's model updates from various attacks is one of the primary core issues in FL. Often, client models can remember individual data patterns, particularly in neural networks. If these model updates are attacked as they are transmitted to the server, deducing the specific client data used to train the model is very plausible [19]. In such cases, it is vital to protect further the privacy of model updates sent to the central server for global model aggregation. We, therefore, introduce the concept of Differential Privacy (DP) [15, 16].

DP has recently received attention since it provides a clear and well-defined mathematical underpinning for privacy. DP algorithms mainly rely on noise injection to prevent an attacker from ascribing the data properties of the clients. The disclosure of sensitive data from the client models can be restricted by employing the different statistical and probabilistic protection techniques available in DP. DP is typically perceived as a randomization technique.

Definition of DP

The defintion for DP is as follows [17, 1], Given two adjacent datasets *d* and *d'*, where *d'* can be formed from by adding or removing all the samples belonging to a single user; a randomized mechanism $M : D \longrightarrow R$ with domain D and range R satisfies (ϵ, δ)-differential privacy if for any two adjacent inputs $d, d' \in D$ and for any subset of outputs $S \subseteq R$ it holds that,

$$Pr[M(d) \in S] \leq e^{\epsilon} Pr[M(d') \in S] + \delta \tag{2.1}$$

The two critical parameters, ϵ, and δ, must be considered while preserving the DP guarantee.

Delta (δ) determines the likelihood that the given privacy assurance may not stand. It determines the risk of sensitive data that will be disclosed inadvertently. As a general rule, it should be less than the inverse of the size of the training data set. Epsilon (ϵ) evaluates the reliability of our privacy assurance. Typically, we desire a small value of epsilon close to zero. However, this is merely the upper limit, and a large epsilon value could still imply ideal privacy in practice in certain use cases.

We can inject noise into the system by utilizing the noise multiplier sigma. The model updates from each client are then wrapped in an additional layer of noise, as seen in Figure 2.3. Ultimately, the FL procedure proceeds and the server aggregates the final model.

We have two main parameters to be considered to design a DP model. One is the gradient norm clipping (C), which ensures a single client's effect on the overall average is bounded to a maximum gradient norm of C. It is a technique for calculating the influence of a client gradient clipping on the overall average (C depends on many factors determined by the model and data). The next is the gaussian noising, which introduces the element of randomness needed by adding $N(0, \sigma^2)$ to a model's parameters (σ is referred to as the noise multiplier).

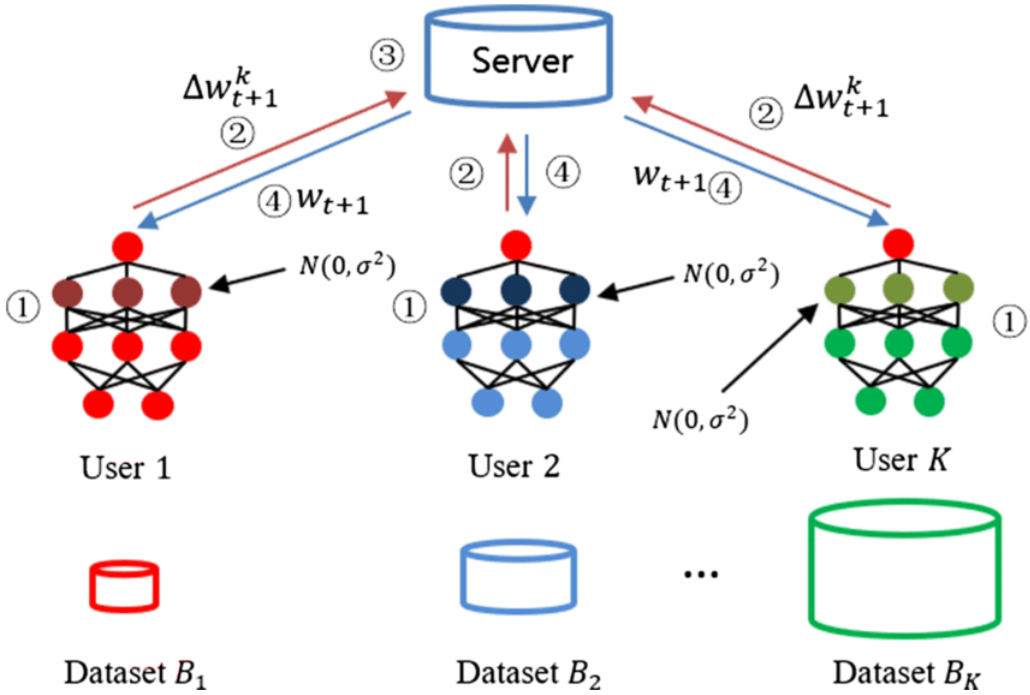

Figure 2.3: The DP-FL framework for the unbalanced data [29]

2.2 LSTM for Time-series Data

Long short-term memory (LSTM) [27, 9], a type of Recurrent Neural Networks (RNN), is commonly employed in deep learning applications. A standard LSTM unit consists of a cell, input gate, output gate, and forget gate. The cell retains values over arbitrary periods, and the three gates regulate data flow into and out of the cell. LSTM, unlike traditional feedforward neural networks, has feedback connections. Thus, it can analyze both single data points and entire data sequences. Therefore, such neural networks are advantageous in situations involving instantaneous data.

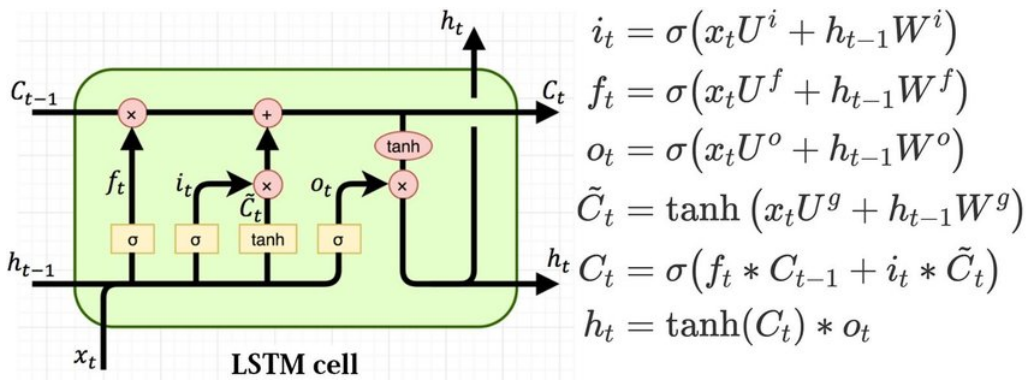

Figure 2.4: Structure of the LSTM cell and equations that describe the gates of an LSTM cell [92]

Figure 2.4 depicts the equations that characterize the behavior of all gates in the LSTM cell [92]. W is the recurrent link between the previous hidden layer and the present hidden layer. Connecting the inputs to the hidden layer is the weight matrix U. $\widetilde{C}$ is a potential hidden state derived from the

current input calculations with that of the previous hidden state. C represents the unit's internal memory, the combination of the previous memory multiplied by the forget gate and the freshly computed hidden state and multiplied by the input gate.

2.3 MLOps Framework for Experiment Tracking in FL

Machine Learning Operations (MLOps) provides the technical basis for ML product lifecycle management. With the help of MLOps, businesses can address many challenges they face by providing user-friendly AI and focusing on critical parameters like automation, scalability, integration, and deployment. Weights and Biases (W&B) is a developer-centric MLOps platform that enables data scientists and ML engineers to create better models and provides experiment tracking, dataset versioning, and model management [6]. To keep track of ML projects and ensure no data loss in the model creation process, W&B serves as the central repository for all relevant data and computation history. Few lines of code are required to accomplish this, signifying its ease of implementation. Additionally, it is ML-framework and cloud platform independent.

W&B's high adaptability, speed, and user-friendliness make it an integral part of the cutting-edge, high-volume research process. It comprises lightweight and interoperable tools. W&B works together on machine-learning projects, maintains an experiment log, and controls different versions of datasets in order to create the most efficient machine-learning pipeline. Findings from individual projects can be monitored and shared with thousands of workers. Further, the in-house roll-out of the ML systems facilitates excellent teamwork.

The key robust attributes of W&B can be summarized as follows,

1. **Rapid Integration and collaboration with the user code**
   Monitoring, contrasting, and visualizing ML trials requires only a few lines of code. It provides a platform for open-source and shareable projects.

2. **Seamless visualization with the interactive dashboard**
   Streaming real-time metrics, terminal logs, and system stats to a centralized dashboard are possible by adding W&B's lightweight integration to the existing ML code. Wanda's multifaceted visualizations make it an interactive and engaging experience for the user.

3. **Real-time results demonstration**
   Typical ML projects involve multiple iterations. Change management is a crucial factor that one should monitor in real-time in such circumstances. In addition, real-time project tracking aids in demonstrating how the model functions, the outcomes of improved versions, the existence of bugs, and the upcoming milestones. Using W&B, we can accomplish all of these.

3 State of the Art

This chapter provides a summary of the literature review conducted for the thesis. Section 3.1 discusses the prior work accomplished in FL in the automotive industry. We use these research works to explain and generate ideas for our FL system development. Using such literature, we discover that the automotive sector requires FL. Section 3.2 explores the various accessible FL frameworks and their merits and limitations to the best of our knowledge. We finally explore the benefits of adopting the chosen FL framework for a production-grade environment and investigate its primary architectural design and components in Section 3.3. Section 3.4 covers the relevant works envisaged for the traffic and mobility simulator. These works help us comprehend the necessity of such simulators for fundamental developmental reasons. Finally, we provide an overview of the insights garnered from these literary works as motivation for the current thesis.

3.1 FL in Automotive Use Cases

The following research discusses the different FL contributions in the field of the automotive industry,

- In this work, [49], an important research contribution that further serves as an inspiration for one of the use cases outlined for the thesis, is discussed. The Fed-BEV framework is a first-of-its-kind framework for the energy modeling of Battery Electric Vehicle (BEV)s using FL approaches. Specifically, a group of BEVs participating in the Fed-BEV framework can collaborate to improve their energy consumption model in a simulation environment by learning from one another. The experimental results indicate that the FedAvg system can successfully improve the prediction capabilities of local models through asynchronous iterations. In addition, a comparative analysis of the usefulness of the suggested framework for accurate energy modeling of BEVs is offered.
- The authors here [79] discuss another vital research contribution in line with one of the use cases defined for the thesis work. The research proposes a novel economic-efficiency framework for EV networks to maximize gains for CSs. Here, a decentralized, federated energy learning (DFEL) approach and clustering are employed for the energy demand prediction of the CSs in Dundee city, the United Kingdom. The implemented approach improves the accuracy with which individual CSs can predict energy demands while significantly lowering communication overhead. The results show that the implemented algorithm can improve energy demand prediction accuracy by up to 24.63 percent compared with other machine learning algorithms.
- In this work, [65], the authors envision a novel approach, Federated Vehicular Network (FVN), as a solution to the issues that arise in uncoordinated mobile/edge nodes. FVN tackles limited storage, computing capabilities, communication bandwidth, and efficiency

loss. Further, FVN aims at the detailed analysis of the routing and caching algorithms that are of predominant use in FL. Further, in this work [99], the authors focus on an industrial, automotive use case. The asynchronous FL improves the prediction performance of local edge models while maintaining the same accuracy as centralized machine learning. The research paper [88] discusses the active research problem, such as resource management and performance optimization in vehicular networks. In another work, [18], the authors examine the various driving applications, such as entertainment and route planning. Several learnings, including data labeling at the edge, model training, communications issues, understanding data rate, transmission overhead, privacy, scheduling, and resource management, are explored in length in this article.

- The research article, [52], presents a comparative analysis of steering angle prediction using a vision-based dataset for both a centralized and a federated machine learning (ML) model for V2X-communicating linked vehicles. The results demonstrate that the suggested FL method achieves the same level of performance as conventional ML with a shorter training time. Using the FL method, it is possible to calculate the other factors influencing the steering angle, such as the brake and throttle settings. The research also indicates that the FL approach can provide a superior alternative to the centralized ML approach on the communication channel by providing a 250-times lighter load under ideal channel conditions and a 62-times lighter load with channel errors, thus consuming less bandwidth.
- The approach presented here proposes an FL method for identifying a bearing fault diagnosis in a specific industrial context [10]. Utilizing the dynamic weighted federated averaging technique protects the confidentiality of user data. The results also reveal that the proposed learning scheme has a greater average diagnosis accuracy than centralized ML learning, suggesting that the former has more potential in practical, industrial settings. Similarly, another work [97] uses transfer learning and federated averaging on shared layers in FL to combine multiple models for the same use case. This is a potential example of federated transfer learning in the automotive industry.

3.2 Open-Source Frameworks for FL

The prevalence of FL's use in AI has prompted the development of several frameworks to accommodate the growing number of possible implementations. Several real-time applications utilize these privacy-protecting frameworks to build a data privacy-protected environment. Multiple criteria, including scalability, flexible and generic software track, supporting agnostic frameworks, communication protocols, support for heterogeneous systems, expandable to production-grade environments, and support for privacy functions, are necessary for forming such frameworks. It is vital to comprehend the benefits and drawbacks of such frameworks to utilize them in future applications. Here, we examine the benefits and drawbacks of several different software frameworks, compare them, and highlight their most significant flaws. Figure 3.1 describes the comparison of the below-explained FL frameworks.

1. **TensorFlow Federated (TFF):** TFF is an open-source framework for machine learning and other computations on decentralized data designed to encourage open research and experimentation with FL [89]. Federated Learning (FL) API and Federated Core (FC) API are the two primary interface levels of the TFF. Under the hood, Tensorflow libraries are used to deploy the federated environment.

   Although TFF fuses effortlessly with existing Tensorflow models and datasets and experimentally with JAX, it is not an ML- and language-independent framework for many other libraries. Therefore, it lacks portability and scalability. Currently, execution is only possible via a local simulation (e.g., using Google Colaboratory notebook with simulated decentralized data), so FL's decentralized architecture is not supported. In addition, the lack of DP mechanisms hinders the ability to implement novel privacy-preserving methods [71].

2. **PySyft:** PySyft [104, 66] is a Python library for secure and private deep learning. It accomplishes this by decoupling private data from model training using the concepts of FL protected with various privacy-enhancing methods and encrypted computation (such as Multi-Party Computation (MPC) and Homomorphic Encryption (HE)). It is compatible with frameworks for deep learning, such as PyTorch, TensorFlow, and Keras. It is a component of the OpenMined ecosystem.

   Due to its low-level implementation and the disadvantages listed below, this framework is complicated and requires extensive expertise in this field to create an FL model effectively. PySyft can only communicate over the network by connecting to clients that contain data and function as servers, limiting its scalability [2]. It does not support datasets by default, making it challenging to test simple instances. There are no model aggregation operators implemented [71].

3. **FedScale:** FedScale [38, 39] provides high-level APIs for deploying and evaluating FL algorithms at scale across various hardware and software backends. It aids in FL tasks on applications such as image classification, object detection, language modeling, and speech recognition. In addition, it provides datasets to simulate FL training scenarios accurately. It currently provides support only for Pytorch and Tensorflow.

   FedScale's lack of actual deployment of the supported multi-machine simulation reduces the system's scalability. Although it can simulate users on numerous machines, scaling is limited to 100 clients concurrently. Its Python-based simulations make them inaccessible for edge devices that may require language-independent behavior.

4. **LEAF:** LEAF is a framework for benchmarking learning in federated environments, with applications including FL, multi-task learning, meta-learning, and on-device learning [8]. This approach is primarily concerned with evaluating FL settings. It contains specific fundamental FL processes, like the federated averaging aggregator, and its modular design is adaptable to any current framework. Moreover, it comes pre-loaded with some well-studied datasets.

   LEAF is limited to single-machine simulations written in Python. It does not establish a standard for preserving privacy in an FL environment, which is an essential factor to consider for FL. In addition, it has less official documentation and tutorials than the other frameworks covered in this area.

	TFF	Syft	FedScale	LEAF	Flower
Single-node simulation	√	√	√	√	√
Multi-node execution	*	√	(√)***		√
Scalability	*		**		√
Heterogeneous clients		(√)***	**		√
ML framework-agnostic		****	****		√
Communication-agnostic					√
Language-agnostic					√
Baselines			√	√	*

Labels: * Planned / ** Only simulated
*** Only Python-based / **** Only PyTorch and/or TF/Keras

Figure 3.1: Comparison of different FL frameworks [2]

In recent years, numerous more emerging frameworks in FL have gained the attention of researchers. Such frameworks include, among others, FedML, FLUTE, Nvidia Flare, and Flower. FLUTE's [13] multi-GPU support and performance enhancements are substantial. However, it is not framework-independent and cannot easily be adapted to a production environment. FedML [25], on the other hand, is production-friendly, although FLUTE is 42x faster and uses 3x less memory than FedML. NVIDIA FLARE^{TM} (NVIDIA Federated Learning Application Runtime Environment) is a reasonably new, open-source, extendable, and domain-agnostic SDK for FL [74]. Consequently, frameworks are evolving in the field of FL, and framework selection is a possible research challenge that can be approached as a separate research topic.

We have chosen the Flower [2, 4, 3] FL framework for the proposed thesis study because of its superior extensibility in a production environment [32, 13]. The adaptable and user-friendly software stack facilitates the ability to adapt our specific use cases. In the following sections, we explore the benefits of selecting this framework and its fundamental architectural components as our planned one.

3.3 Overview of Flower Framework

Flower is an open-source, scalable, production-friendly, end-to-end FL framework for establishing FL-based client-server communication [2]. It finds applications in various fields, including medical, automotive, production-ready, mobile, and wireless clients. Flower's steadfast software stack and higher abstraction level enable researchers to develop new applications quickly.

Flower provides built-in tools for simulating many challenging conditions in a cloud environment and enables a realistic evaluation of their impact. It is also possible to seamlessly transform the existing centralized training pipeline into a federation environment achieving privacy goals. Due to Flower's high scalability, it can accommodate thousands of users. It supports single-machine core simulation or across a distributed cloud server network, providing more straightforward simulation options. It supports realistic system conditions by allowing training and evaluation on edge devices and single and multi-node clusters. Additionally, it supports the distribution of heterogeneous data,

paving the way for scalable algorithmic research into real-world system conditions like constrained computational resources, which are typical for FL workloads. It is a stable framework where we can implement language and ML framework-agnostic algorithms, thus addressing different challenges in FL. The source code for Flower is available in the GitHub repository [20].

Flower attempts to tackle the shortcomings of the various frameworks in the preceding section. It offers a simulation environment, is independent of ML frameworks and programming languages, and supports secure aggregation [40] to ensure that the server cannot inspect individual trained models during aggregation and the implementation of new differential private algorithms, to name a few. It also supports on-device FL training [54] for various smartphones and devices. It is also scalable, and additional tools may be constructed on top of Flower; for instance, the resource-aware simulation client profiling tool in this study is named Protea [100].

The fundamental primitive contributions from the Flower framework support both the system-level as well as algorithmic approaches as listed below,

- **Scalability to large-cohort training and evaluation of clients:**
  It supports the large-scale training and evaluation of clients on real-time edge devices and on single-node or multi-node compute clusters. Thus this enables scalable pool-size of clients for researchers to better understand and propose design concepts for FL. Further, it unveils the complexities of resource allocations in computation-intensive tasks. Flower supports client scaling up to 15 million [4].
- **Agnostic Framework:**
  Providing an agnostic framework scales multiple technologies' adoption and implementation. Flower framework provides the following agnostic features.
  - ***Client agnostic:***
    Most of the data from real-world scenarios involve non-identical heterogeneous environments. Such heterogeneous experiments can be easily deployed on Flower, thus adapting to a more versatile data environment.
  - ***ML framework and language agnostic:***
    The heterogeneous environment of the client naturally leads to the adoption of different ML frameworks and workloads. Examples range from situations where clients use two different training frameworks (PyTorch and TensorFlow) to more complex situations where clients have a device- and OS-specific training algorithms.
  - ***Communication-agnostic:***
    Given the heterogeneous connectivity settings, Flower allows different serialization and communication approaches.
  - ***Privacy-agnostic:***
    Different FL settings have different privacy requirements, like secure aggregation and differential privacy. It supports these common approaches while not being rigid about their usage.
- **Flexibilty:**
  Flower provides a smooth transition between virtual testing and real-world implementation. It is flexible to enable experimental research by providing numerous design aspects and FL

algorithm support. It can be easily adapted to new methods with minimal overheads. Flower offers both single-node and multi-node simulation support, with simulations as a genesis for innovations.

Core Architecture Design

Setting up an FL environment requires configuring workloads on both the client (local) and server (global) sides. Each client has its dataset, model, set of hyper-parameters, training, and evaluation methods, which may or may not be the same for all clients in FL. Each client's local computation is tailored to their specific needs. The individual local training pipeline of each client handles their client-specific modifications. The server-side global computation orchestrates the entire learning procedure and client-server communication. It is oblivious to the information stored on the client side. Additionally, clients are unaware of one another's data. The server's primary function is to implement a Strategy to exercise and coordinate the training and evaluation phase in FL.

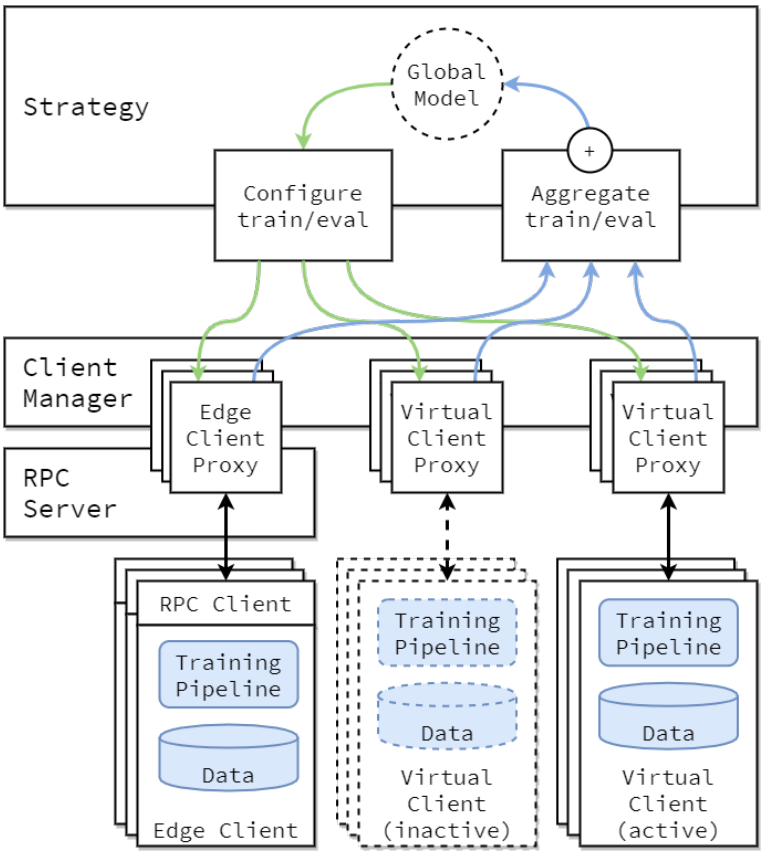

Figure 3.2: Flower Architecture - Edge Client Engine and Virtual Client Engine [2]

The core architecture of the Flower framework is depicted in Figure 3.2. The key elements include the training pipeline of the edge client or virtual client, the RPC server for the edge client, the client manager, and Strategy. The framework allows simulation and real-time testing for virtual and edge clients. The edge clients represent real edge devices and communicate over RPC with the server. On the other hand, virtual clients are used for simulation purposes. An instance of the class is

created for the virtual clients only when they are selected during the FL process. Virtual clients consume nearly zero resources when not participating in the FL process. We only consider the operation of the virtual clients for the use cases described in the upcoming sections. Virtual clients represent the EVs and CSs in our study.

VCE

Virtual Client Engine (VCE) is a tool that facilitates the virtualization of Flower clients in order to maximize hardware utilization. Clients will be scheduled, instantiated, and run by the VCE in a completely invisible manner to both the user and the Flower Server. During the FL iterations, the VCE manages the resources the clients require in a resource-aware manner, including the number of CPUs, GPUs, and VRAM requirements.

VCE achieves the objectives listed below:

- Process and task parallelization for a large client base
- Avoiding over and under-utilization of the memory resources
- Offering an interoperable and portable platform for FL experiments across configurations without the need for reconfiguration
- Minimal overheads
- Large-scale cohort simulation

Server Components

The server-side design mainly entails the following components:

1. **User-customizable Strategy**
   Flower enables user scenario-specific customization by providing an interface called Strategy Abstraction. We can customize an existing strategy with callback functions. By deriving from the base class implementation in Flower, we can implement user-defined custom strategies. Several built-in strategies are provided in the core framework. Strategies here regulate the order of actions, namely client selection, algorithm implementation, client configuration, and result aggregation for training and evaluation.
2. **FL loop**
   The server's very architecture relies on the FL loop. The FL loop orchestrates server-client communication utilizing the Strategy implementation. It acts as a bridge between the ClientManager and the Strategy. It does this by sending configuration parameters to the clients, getting updates (or failures) from them, and letting the Strategy handle aggregating the obtained results. Since FL is an iterative approach, the FL loop requests the Strategy to set up the clients for the subsequent iterations.

3. **ClientManager (Client selection and management)**
   The management of the client pool is an integral part of the architectural design. The server handles this through the ClientManager component. The ClientManager manages a collection of ClientProxy objects created for each client participating in the FL process. ClientProxy is only created for clients connected to the server during training or evaluation. They are in charge of transmitting and receiving Flower Protocol messages to and from the actual client.

3.4 Usage of the SUMO Simulator

Gathering EV-driving data can be a strenuous process that involves concerns about client privacy and a need for more interest among clients in sharing their data. In such scenarios, a meager amount of data accumulation affects the development and evaluation of AI algorithms for automotive use cases. Simulators contribute significantly to the data needed to develop the initial EV-AI evaluation use cases by simulating the various characteristics present in real-world systems and providing an abstraction for diverse architectures [26].

Eclipse' Simulation of Urban MObility (SUMO) is an open-source, highly portable, microscopic, and continuous traffic simulation package developed by the Institute of Transportation Systems at the German Aerospace Center [51, 94, 21]. The simulator can handle large complex networks allowing inter-modal simulation. It features flexible tools across multiple traffic and mobility design generations. Many fields, including traffic engineering, mobility simulation, and transportation management, have found a use for this well-validated simulator. Features included here, like electric vehicle mobility patterns, speed limit distribution, lane change rules, and car following models, make the simulation as close to reality as possible. In addition, SUMO can help us calculate the potential energy consumption of EVs and the charging infrastructure.

SUMO is acknowledged as one of the most advantageous traffic and mobility simulation software. SUMO's diverse toolbox enables us to create a variety of real-time traffic simulation scenarios. Further, it is possible to adapt and map the attributes from such simulation tools to real-time data assembling to get near-probable results. The following studies employ SUMO as their primary simulator for designing such use cases.

- The authors in this work [53] extend the SUMO 2D vehicular simulation into a 3D simulation mode, with components affecting EV performance, such as terrain slope, directly influencing the pace at which the battery discharges over time. The improved model will help investigate more energy-efficient route selection for EVs in 3D environments. The simulation design used to implement the improved model helps us understand the possibility of adapting the simulation steps according to the user's needs. Thus the modifications outlined in the paper for the enhanced large-scale electric transportation SUMO framework indicate that it is flexible enough to accommodate new configuration models for EVs.
- In this work [22], the authors have presented one of SUMO's large-scale and high-accuracy traffic simulation scenarios that are openly available [80]. The scenario accurately models the traffic flow, speed, and road occupancy for nine days over a 97 km freeway between Alicante and Murcia in Spain. It is a valuable testbed for Vehicle to Everything (V2X), Connected and Autonomous Vehicles (CAVs), and Intelligent Transportation Systems (ITS) research and

engineering. The design concepts presented here assist us in comprehending the calibration approaches and diverse toolkits available within SUMO that contributed to the dataset's creation.

- The authors in this work [91] evaluate the impact of manual driving and dynamic-flexible platooning on battery energy consumption using the SUMO EV energy model. This work also reveals the effect of speed and acceleration on battery energy consumption.
- In this work [90], the authors describe the design setup for simulating a BEV in the SUMO simulator on a simulation basis. In addition, the accuracy of the energy estimation model for the battery SOC in SUMO is compared to real-world experimentation by driving a BEV between St. Pölten and Linz in Austria.

The increasing prevalence of EVs on the road necessitates resilient, scalable service solutions that can adapt to new circumstances. However, the lack of comprehensive and adaptable real-world EV data sets remains a barrier to the widespread adoption use case evaluations on EVs. The SUMO literature review above shows that it is a highly scalable traffic simulator with many features to model the required traffic scenarios. We can thus improve adaptability, efficiency, and visibility by setting up many test beds for various ecosystems using the SUMO simulator.

3.5 Derived Insights

From the preceding, it is clear that the employment of FL in the automotive industry ushers in a revolutionary change in how user privacy is handled. We also recognize its expertise in a variety of use scenarios. We also observe that its performance is comparable to conventional machine learning techniques. In addition, numerous frameworks are available for implementing FL, of which we choose one that satisfies all the essential aspects of our use cases. Finally, we acknowledge the significance of employing simulators in related works to construct a realistic environment in the face of limited data.

Driven by the preceding observations, we focus primarily on the three contributions required to answer the Section's research questions.

- Formulation of automotive use-cases
- Developing a realistic simulation test bed with the SUMO simulator to simulate EVs and CSs
- Leveraging and modifying the Flower framework to accommodate the use cases

4 Methodology

This chapter outlines the two use cases highly prevalent in the EV automotive industry to address energy consumption and power demand. The methodology proposed explains the design establishment of an FL environment to address the two discussed use cases. The chapter is further organized into the following sections.

4.1 **Proposed Use Cases**
Use-case (a) and Use-case (b) are defined here.

4.2 **Data Generation**
Here the simulator environment and tools responsible for the EV and CS data acquisition is discussed. The mathematical notations of the simulator's energy model are further elaborated. Finally, the entire simulation setup used for the work is discussed in detail.

4.3 **FL Settings**
We discuss adopting the Flower FL framework for establishing client-server communication for the use cases. The different algorithms and strategies used as a backdrop in the design concept to realize the two use cases are detailed here.

4.4 **Weights and Biases**
The developer-friendly MLOps platform integrated with the proposed FL system for experiment tracking is explained here.

4.5 **Employed Evaluation Metrics**
The important regression metrics that are used for benchmarking the prediction and forecasting results for the battery-capacity of the EV and energy demand for the CS are outlined mathematically here.

4.1 Proposed Use Cases

In this work, we propose two scenarios for applying FL in an automotive setting, each of which sheds light on the computing paradigm's utility and operation. We use SUMO [26] for EV and CS model simulation and data acquisition. We use Flower federated framework [2] to set up the FL environment. The upcoming sections will discuss the simulator and framework in detail.

4.1.1 Use-case (a) : Actual battery capacity prediction for EVs

The various SUMO simulator tools and SUMO's existing energy model are utilized for modeling and simulating EV mobility. The region, types of EVs used, and the characteristics of the individual EVs determine the nature of the generated data. The EV's battery capacity during mobility is recorded during the simulation. We build the FL system with the available instantaneous data for each EV.

The central server gathers model updates from all participating EV clients. After aggregating the local updates, the server returns a global model to the evaluation clients. The training and evaluation of the configured clients occur with a federated backdrop strategy using Flower's design concept. The various parameters, such as the local client's model, FL hyper-parameters, client selection strategy, and assessing model performance, are other vital aspects examined in the FL setting. Thus, FL protects data privacy by utilizing federated-iterative communication between the server and clients to train models on private EV client data. Every client receiving the updates further determines its battery capacity by deploying the new global model updates. The training continues until a convergence rate is met. Additionally, we provide end-to-end privacy guarantees for each client using the differential privacy concept here (i.e., each EV client has a DP model). Figure 4.1 illustrates the overview of this use case.

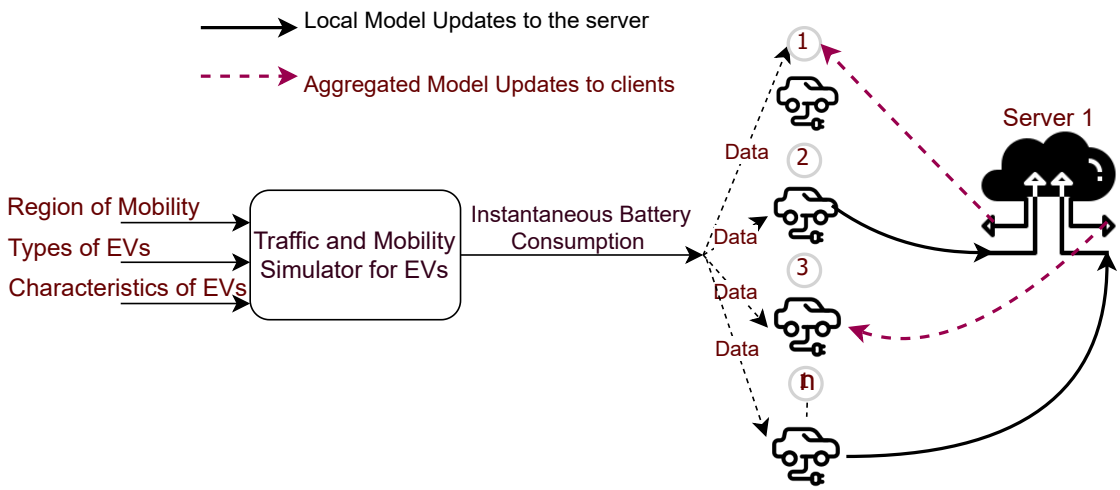

Figure 4.1: Use-case (a): Actual battery capacity prediction for EVs

4.1.2 Use-case (b): Energy-demand forecasting for CSs

To assess the energy usage of CSs, we install them in multiple locations using the SUMO simulator's netedit tool. Furthermore, the energy consumption of CSs is monitored when EVs use them. Such CSs train their data locally and send the model updates to the central server. The global model updates are then transmitted to the requested CSs, and the CSs analyze their respective energy demand for their test or unseen data. In this use case, we also deal with defective clients to

demonstrate and design a fault-tolerant system. When a CS chosen for training or evaluation has insufficient data samples to be trained, we drop such clients from the process. Further, a notification is sent to these clients. Figure 4.2 illustrates the overview of this use case.

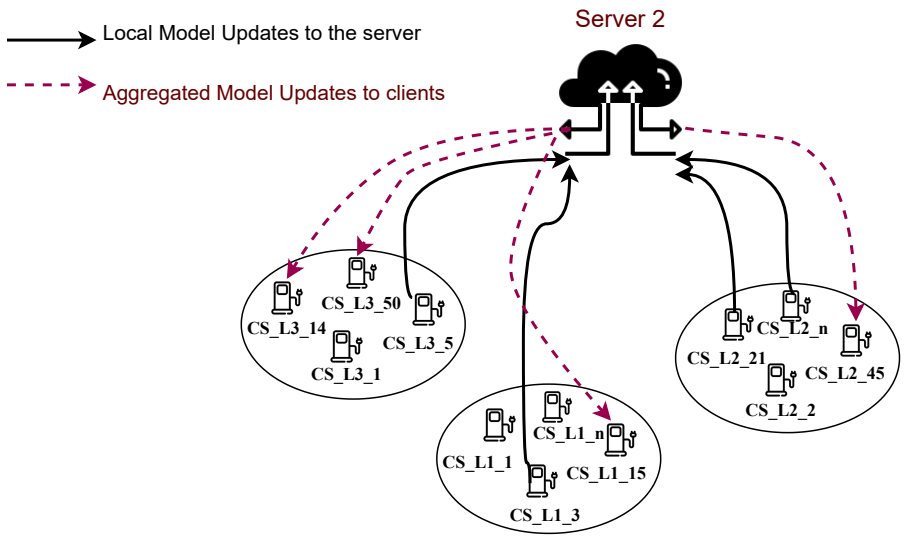

Figure 4.2: Use-case (b) - Energy-demand forecasting for CSs

4.2 Data Acquisition and Modeling

The mobility of the EVs affects the CSs when the EVs use them, leading to an inter-linking traffic and energy demand simulation model. As discussed previously, we set up a simulation test bench to address the data acquisition issues using SUMO, a microscopic traffic simulator. The factors impacting battery capacity throughout various driving cycles can be analyzed using SUMO.

The overall energy model for EVs comprises the subsystems depicted in Figure 4.3. SUMO's EV energy model is a straightforward representation of the runtime EV energy consumption calculations based on the mechanical subsystem. During modeling the mechanical subsystem, the force applied to the wheels during the movement of an EV is reflected. Here, the EV energy consumption is modeled with the vehicle dynamics equation, primarily dependent on speed, acceleration, rolling resistance, aerodynamic drag, and road slope forces. Here, the mechanical subsystem is considered the most influential aspect of the EV energy consumption model [75, 76].

SUMO streamlines the electrical subsystem by adding a constant efficiency propulsion parameter that the EV constructor supplies. This parameter represents the additional electric loss that arises from the battery and dissipates as the car accelerates. The auxiliary energy losses are represented by a constant power provided by the vehicle manufacturer. In the proposed model, the recuperation part of energy consumption is represented as a constant regenerative braking efficiency factor provided by the vehicle manufacturer.

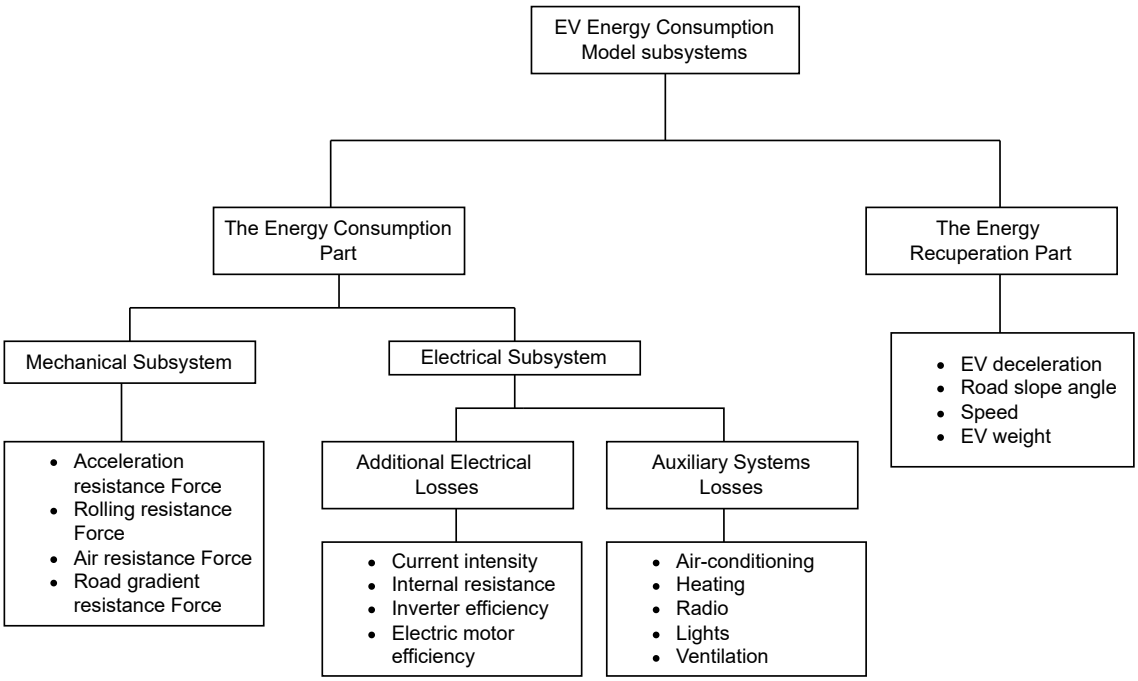

Figure 4.3: EV energy model main subsystems [75]

4.2.1 EV Energy Model in SUMO - A Mathematical Representation

SUMO is one of the simulators that employ backward-facing energy models [7] to calculate a vehicle's energy consumption. As T. Kurczveil et al. [37, 77] proposed, the energy model in SUMO can provide instantaneous energy consumption estimates with only a handful of vehicle-specific characteristics. They maintain computational efficiency by deriving the consumed energy using simple equations that relate the current speed to the energy supply.

We outline the consumption model that we used to simulate the energy consumption of EVs. The SUMO traffic simulator's model, suggested by Kurczveil et al., serves as the model's mathematical foundation. It is necessary to ascertain each vehicle's energy to estimate energy consumption. This can be calculated by adding the kinetic, potential, and rotational energies and deducting the energy the vehicle's parts expended. The following equation 4.1 demonstrates how the model gauges a vehicle's energy at a given instant k:

$$E_{veh}[k] = E_{kin}[k] + E_{pot}[k] + E_{rot}[k] = m/2 \cdot v^2[k] + mgh[k] + J_{int}/2 \cdot v^2[k] \tag{4.1}$$

where m is the vehicle's mass, v[k] is the time variant speed, g is the gravity acceleration, h is the time variant vehicle altitude, and J_{int} represents the moment of inertia of internal rotating elements.

After determining the energy of the vehicle, we calculate the energy spent or restored between instants k and k+1, including the energy loss as in equation 4.2:

$$\Delta E_{gain}[k+1] = E_{veh}[k] - E_{veh}[k+1] + \Delta E_{loss}[k] \tag{4.2}$$

To determine the total energy loss, we add up the energy lost due to air resistance, ground resistance, curve resistance, and the continuous energy consumed by each vehicle.

$$E_{loss}[k] = E_{air}[k] + E_{roll}[k] + E_{curve}[k] + E_{const}[k] \tag{4.3}$$

Calculating the various parameters of the equation 4.3 is as follows:

$$\begin{aligned} \Delta E_{air}[k] &= \frac{1}{2}\rho_{air} \cdot FSA_{veh} \cdot Cd \cdot v^2[k]|\Delta s[k]| \\ \Delta E_{roll}[k] &= c_{roll} \cdot m \cdot g \cdot |\Delta s[k]| \\ \Delta E_{curve}[k] &= c_{rad}\frac{mv^2[k]}{r[k]} \cdot |\Delta s[k]| \\ \Delta E_{const}[k] &= P_{const} \cdot \Delta t \end{aligned} \tag{4.4}$$

where ρ_{air} is the air density in the simulated scenario, FSA_veh is the front surface area of the chosen EV, Cd represents the vehicle air drag coefficient, s[k] is the distance covered by each EV, c_{roll} refers to the rolling resistance coefficient, c_{rad} is the radial drag coefficient, and P_{const} is the constant power intake.

If the change in vehicle energy between instants k and k+1 is positive or negative, it is multiplied by a propulsion or recovery factor based on how the batteries charge or discharge. We calculate the level of the vehicle's battery at instant k + 1 using the equation 4.5 and 4.6.

$$E_{bat}[k+1] = E_{bat}[k] - \Delta E_{gain}[k] \cdot n_{prop} if \Delta E_{bat} < 1 \tag{4.5}$$

$$E_{bat}[k+1] = E_{bat}[k] - \Delta E_{gain}[k] \cdot n_{recov} if \Delta E_{bat} > 1 \tag{4.6}$$

where, n_{prop} refers to the propulsion factor, and n_{recov} refers to the constant recovery factor.

Depending on its sign, $\Delta Egain[k]$ represents the amount of energy consumed or regained by the vehicle as a result of its motion.

4.2.2 Traffic and Mobility Simulation in SUMO

Electric Vehicles

The electric vehicle is implemented as a device in the SUMO package. The added device encapsulates the functionality for vehicle types known as vType [84]. The implemented device computes the electric power consumption at every time step. Further, it also checks the vehicle driving state's validity to detect operating points beyond the limits of the motor. Figure 4.4 outlines the simulation setup used here.

The simulation in SUMO comprises the utilization of the following toolset. The tools manifest several pre-processing techniques to construct a feasible network for the EV mobility simulation.

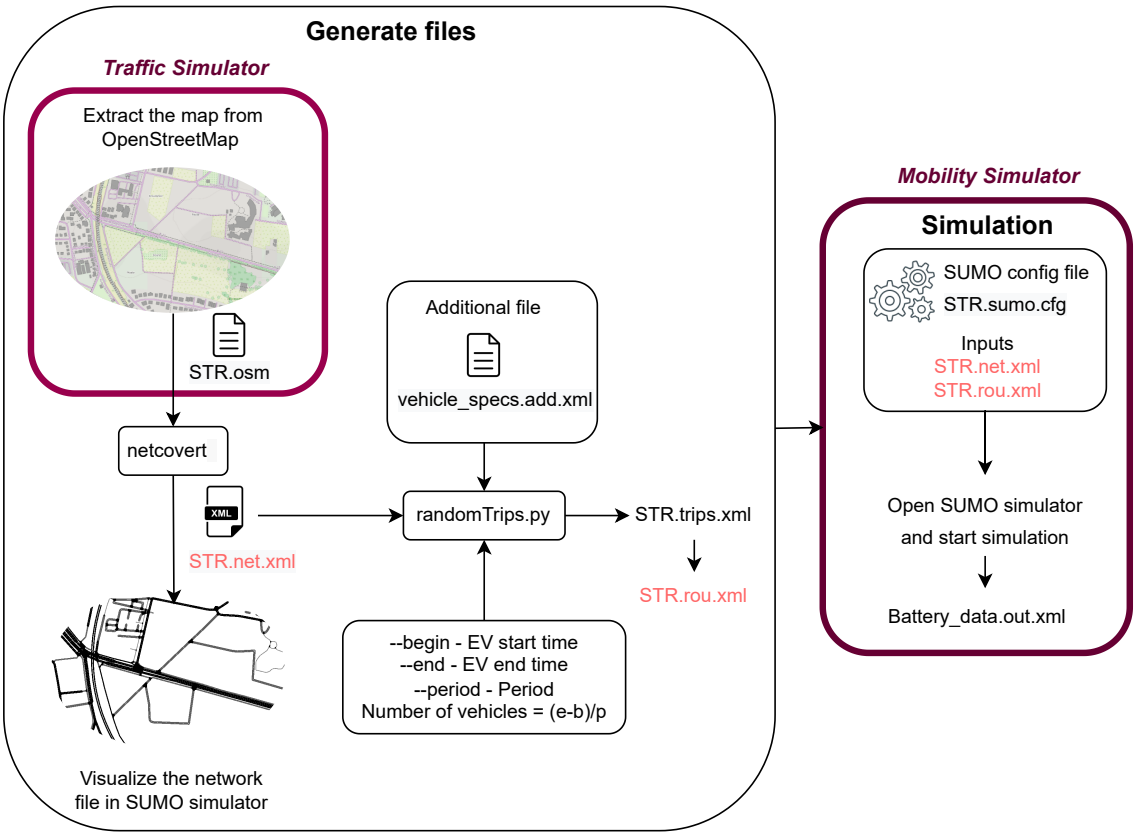

Figure 4.4: SUMO Set-up - Traffic and mobility simulation.
The different CLI tools in the SUMO simulator are used to generate the required output.

Table 4.1: Simulation Tool Versioning

	Description	Version
Simulator	Eclipse SUMO	1.14.1
Netedit	Netedit	1.14.1
Map Generator	OpenStreetMap	

1. **OpenStreetMap**
   OpenStreetMap (OSM) is a collaborative effort to build a free, editable global geographic database [62]. As a part of the traffic simulator, we use OpenStreetMap to acquire regional geographic data from a marked area of interest. The resulting file is in the .osm format. Figure 4.5 illustrates a sample map extracted from OpenStreetMap.

2. **netconvert**
   A command line utility tool, netconvert from the SUMO simulator, imports the map data from the osm file and converts it into a SUMO-readable network XML file (net.XML) [59]. Figure 4.6 illustrates the corresponding netconvert converted map for the .osm file in Figure 4.5.

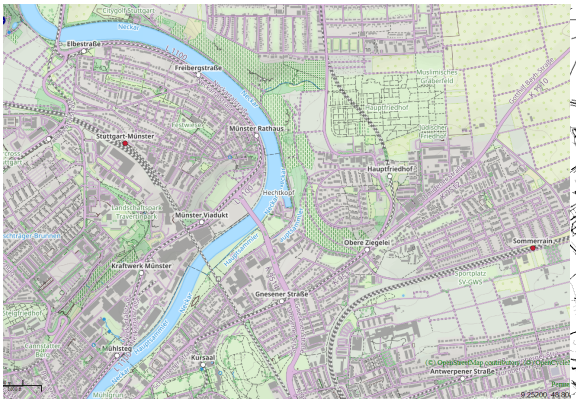

Figure 4.5: Extracted map from OpenStreetMap

Figure 4.6: netconvert converted map

3. **Additional files**
   The next step is to configure the characteristics of the EVs. For this purpose, we use an additional XML file (add.XML). The input attributes considered for these vehicles to obtain the battery capacity are vehicle speed, acceleration, and battery capacity. We specify the maxSpeed, maxAcceleration, and maxbatteryCapacity in the additional file for each used EV. The –trip attributes are used to pass the vehicle type from the additional XML file. SUMO identifies distinct EV features in the GitHub repository [21] utilized in the subsequent study [35]. We utilize similar attributes for the setup of the simulation later.

4. **speedDev, speedFactor**
   To further utilize the maxSpeed during the EV mobility, we assign a speedFactor to individual vType. The product of the road speed limit and speedFactor results in the speed with which the EV commutes in a given area. When the speedFactor is greater than 1, the EVs travel with a speed more significant than the edge speed. Setting the speed distribution using speedDev overrides the constant speed, thus resulting in a diverse profile of speed and acceleration values. This mode is mainly used to simulate heterogeneous mobility patterns.

5. **randomTrips.py**
   The randomTrips python utility script renders a set of random trips for a given network [86]. The generation of trips between the source and destination points follows as per the details specified by the user. We store such generated trips in the trips.XML file. Using the start, end time along with the period specified during the simulation, we can determine the number of vehicles in the current simulation setup using the formula below,

   $$EV_{total} = \frac{e - b}{p} \tag{4.7}$$

   where e - end time for the simulation in seconds
   b - start time for the simulation in seconds
   p - period of occurrence of the EVs

6. **duarouter**
   The duarouter from SUMO computes the shortest or optimal path in the specified network [85]. The routes between the start and the endpoints are stored in the rou.xml file.

Figure 4.7 illustrates the EVs' mobility in different lanes in the SUMO simulator. They follow a car following model in SUMO and obey the necessary traffic conditions (tls).

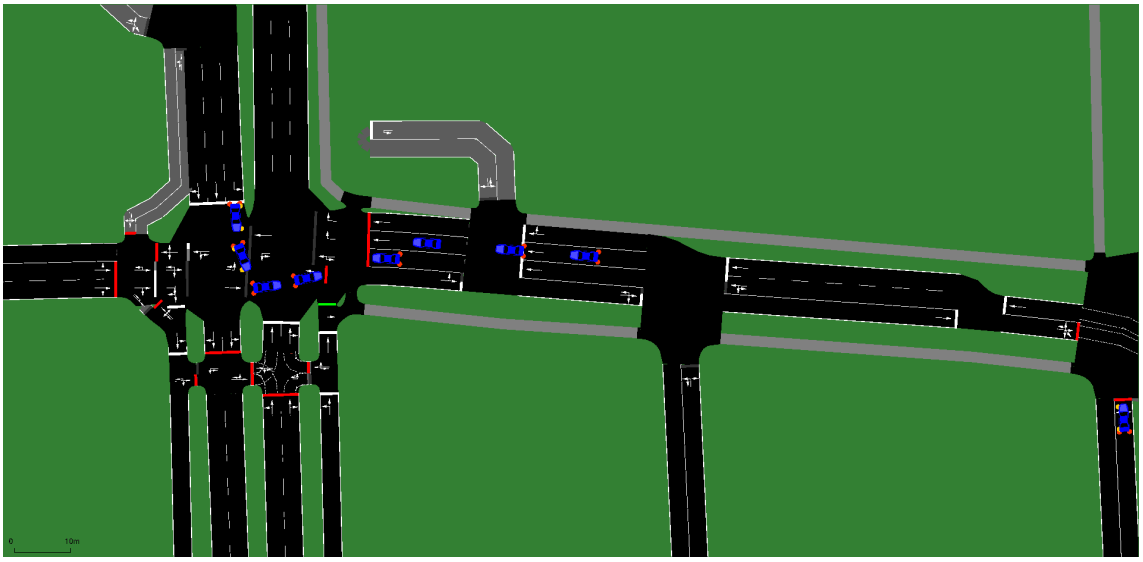

Figure 4.7: EVs during the simulation in SUMO

Charging Stations

netedit

netedit is a graphical editor for networks [60]. We import the net.XML map from the netconvert utility above using the netedit tool to install CSs. It may be used to build networks from scratch and adjust all features of existing ones.

We can charge the battery-powered vehicles using the installed CSs in the locality (here, Stuttgart region) using the netedit tool. Figure 4.8 illustrates that the lane with the installed CS changes its color from blue to yellow to indicate that an EV is using this particular CS for charging.

To install a CS in a valid lane, we need to specify the following attributes [84] as in Figure 4.9, for each CS using the additional tool in netedit,

- **id:** Specify the CS ID (CSID)
- **name:** Enter the name of the CS
- **friendlyPos:** When this option is enabled, the detector will not indicate an error if the CS is placed behind the lane. Instead, the detector will be positioned 0.1 meters from the lane's end or at 0.1 if the position is negative and more significant than the lane's length after multiplication with -1. Setting this is particularly useful in an extensive network where an educated guess is required to install the CS. Here we enable this option during CS installation.
- **power:** We use the most common type of CS with up to 22 kilowatts of power widely used in Germany.
- **efficiency:** We set it to 0.95 as the default for the simulation.

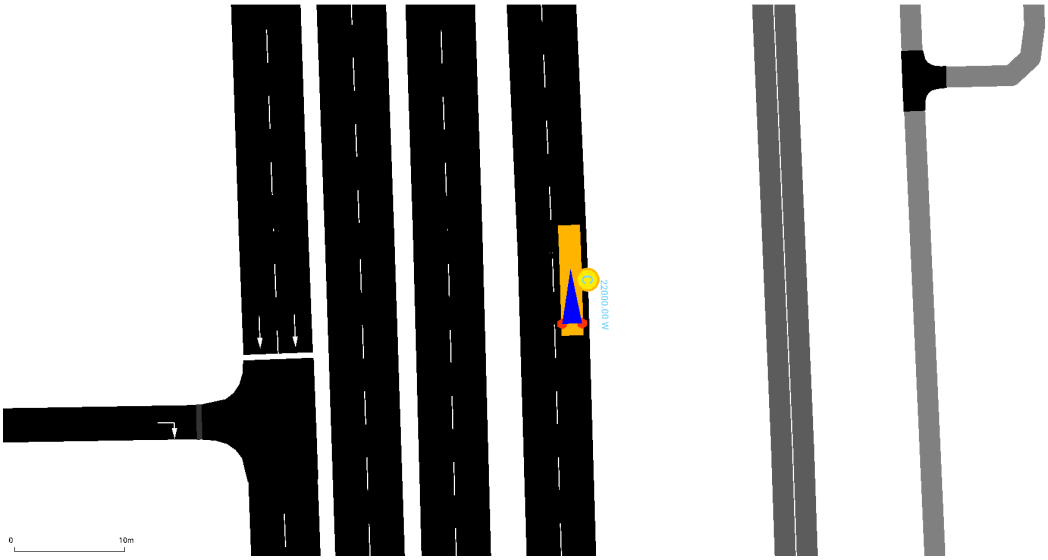

Figure 4.8: Electric Vehicle using a CS to recharge

- **chargeinTransit:** This option enables or disables charge in transit, i.e., the vehicle is forced/not forced to stop for charging. We enable this option for the simulation here.
- **chargeDelay:** Time delay after the vehicles have reached/stopped on the charging station, before the energy transfer (charging) starts. Currently, this is assigned the default value of 0 for the simulation.

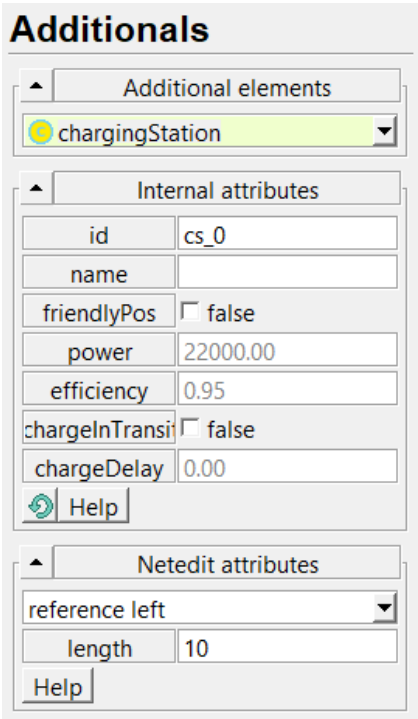

Figure 4.9: Attributes of a CS during its installation

4.3 FL Settings

The Section 3.3 provides an overview of the core Flower federated framework architecture and its benefits. Here, we describe in depth the design adaptation of the framework to our automotive use cases.

4.3.1 Strategy - FedAvg and FaultTolerantFedAvg Algorithms

FedAvg

Federated Averaging FedAvg is a crucial and widely used algorithm in FL that was initially introduced by McMahan et al. [55]. It is an aggregation algorithm specifying how each local model parameter is aggregated at the server while maintaining privacy. It aggregates and averages the results of SGD performed locally at each client level by a central server. The central server returns this global model to each client, and this process is repeated until achieving the required convergence rate.

Algorithm 4.1 FederatedAveraging [55]. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

Server executes:
  initialize w0
  for each round t=1, 2,..... **do**
    m $\leftarrow max(C.K, 1)$
    $S_t \leftarrow$ (random set of m clients)
    for each client $k \in S_t$ **in parallel do**
      $w_{t+1}^k \leftarrow$ EV_ClientUpdate(k, w_t)
      $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} \cdot w_{t+1}^k$

EV_ClientUpdate(k, w): // Run on client k
  $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches size B)
  for each local epoch i from 1 to E **do**
    for batch $b \in \mathcal{B}$ **do**
      $w \leftarrow w - \eta \triangledown l(w; b)$
  return w to server

We employ the Flower server configured with the FedAvg strategy with enhancements for the actual battery predictions for the EVs, as explained in Use-case (a). A generalized FedAvg algorithm is as described in Algorithm 4.1. Federated Averaging works in the following steps:

1. When the FL process begins, an EV client is selected at random. The central Flower server initializes a set of weights w0. These weights are then relayed to individual EV clients.

2. As the FL round begins, using the client-picking strategy in Flower (explained in the following section), clients selected for training use SGD to train their local models using their respective client data. After the first round of training, a set of local model gradients is obtained from these clients.

3. The previously determined local model gradients are then transmitted to the central Flower server without sending the actual client data. The server aggregates the gradients, which are then used to update the global model weights via a weighted average of the gradients.
4. Again, the client-picking strategy is used to send the global model weights to the selected evaluation clients.
5. The above process repeats until the user-specified number of FL rounds.

FaultTolerantFedAvg

FaultTolerantFedAvg is a modification to the FedAvg algorithm that handles fault tolerance in the system. FaultTolerantFedAvg completes an FL round when faulty clients are encountered. In FaultTolerantFedAvg, we specify two critical parameters,

- **min_completion_rate_fit** - A fraction of clients required to generate results for each FL training round.
- **min_completion_rate_evaluate** - A fraction of clients required to receive the results for each FL evaluation/testing round.

We can describe the two parameters mentioned above using CSs' example. Suppose we sample 100 CS clients in the initial round of FL to determine the energy demand in a location. Consider that we had 20 faulty clients with insufficient or no training data. The training ceases when the system encounters errors due to inadequate client training samples. The failures are evident in the logs generated by the FL training procedure. We also view the number of successful and unsuccessful client outcomes in these logs. Based on this input, we implement the FaultTolerantFedAvg in the system. By setting min_completion_rate_fit to 0.8, we will receive an aggregate of 80 client results and ignore the failures from the 20 faulty clients. Further, a notification is sent back to the faulty clients to resolve their issues. These parameters are implemented in a user-customizable manner to adapt them to every FL training round.

The Figure 4.10 illustrates the working of FaultTolerantFedAvg with minimum completion rate for training and evaluation.

4.3.2 Flower Methods for Strategy Implementation

The four essential methods used in the FedAvg algorithm to aggregate and send the results from and to the clients are as follows,

- **configure_fit method**
  It is responsible for planning the next round of training. Configuring a round involves selecting clients and determining instructions to be sent to the clients for training.
- **aggregate_fit method**
  It is responsible for aggregating the results returned by the clients selected and requested for training in configure fit.

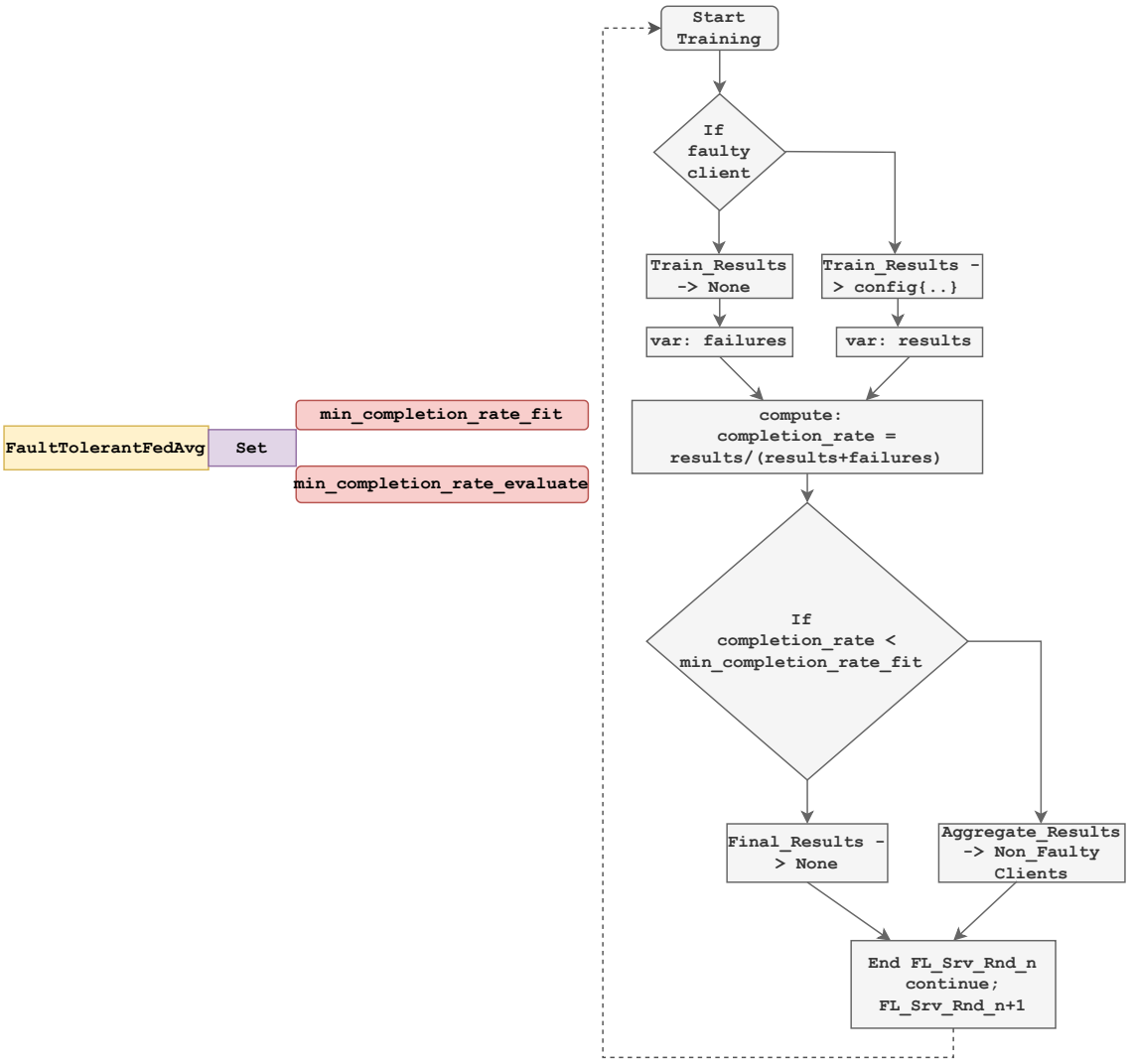

Figure 4.10: Flow chart - FaultTolerantFedAvg

- **configure_evaluate method**
  It is responsible for designing the upcoming evaluation round. Configuring a round involves selecting clients and determining the instructions that need to be sent to them.
- **aggregate_evaluate method**
  It is responsible for aggregating the results returned by the selected clients that configure evaluate requests to evaluate

4.3.3 Client Picking Strategy

The client selection here is based on querying the user-furnished client IDs. The user is prompted to enter the EV client IDs required for the training and evaluation process when the FL process begins. The selected training clients undergo training and send the model updates to the central server. The aggregated updates from the global model are then sent to the clients that have been chosen.

We alter the FedAvg strategy to enhance and implement the client-picking strategy for training and evaluation. For this purpose, we implemented a new class called ClientPickingStrategy. The implemented class overrides the Criterion abstract base class in Flower server. The select() method of this class selects only the specified clients, as illustrated in Figure 4.11. Thus the implemented strategy helps us decide the clients to be used for both use cases involving the EVs and CSs. The aforementioned implementation demonstrates that the Flower framework easily adapts to the user's specifications.

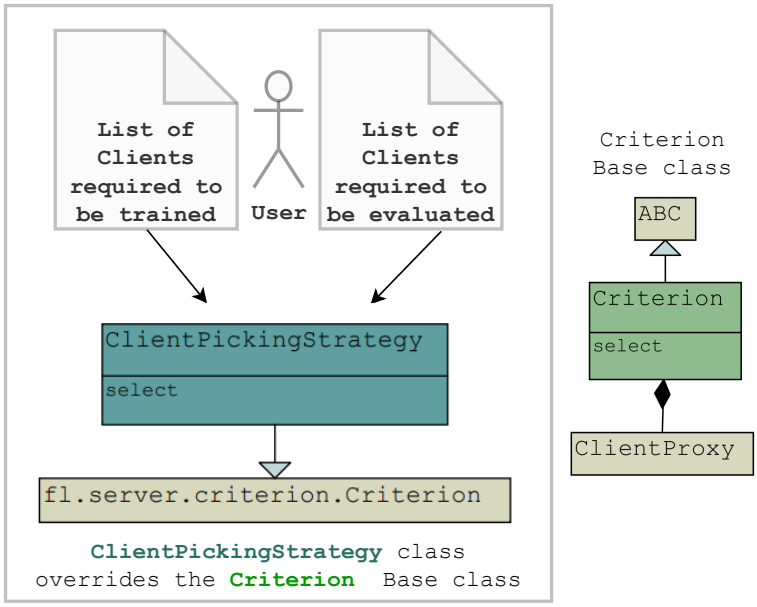

Figure 4.11: Client Picking Strategy

4.3.4 Client Manager

Flower offers a resource-efficient abstract base class, ClientProxy, that enables client interactions independent of the client's identity. It is efficient regarding resources because it is created only for clients selected for training and evaluation. Each created ClientProxy returns the individual client configurations, accounting and receiving client parameters and client results, and finally enabling communication with each client. No ClientProxy object is associated with disconnected clients.

The abstract methods in the ClientProxy class provide the following usage. Each client ID is represented as cid. The client properties are returned by the get_properties() method in Flower. The get_parameters() method returns the local model parameters currently in effect. The fit() and evaluate() methods further refine and evaluate the provided parameters using the locally held dataset.

Finally, the reconnect() method is used to disconnect or reconnect the clients. A UML diagram,

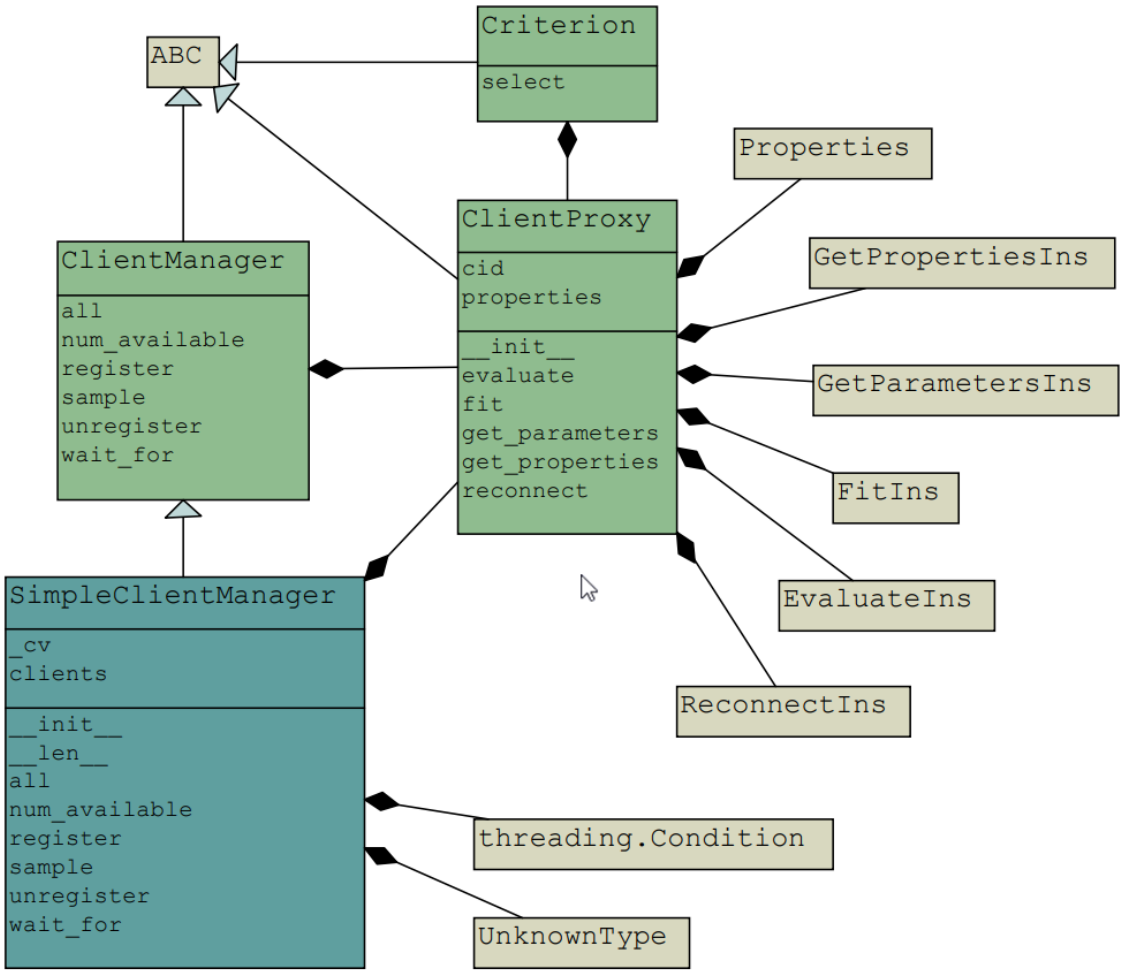

Figure 4.12: UML Diagram - ClientManager and ClientProxy

as in Figure 4.12, explains the usage and connection between the ClientProxy and ClientManager abstract classes in Flower.

ClientManager is an abstract base class that provides several methods for managing clients. The num_available() method returns the number of clients available during each round of FL training and evaluation. The register() method is used to register these clients. The sample() method is then called to sample clients. It returns a list of the ClientProxies that were requested.

When a criterion is used, as described in the client-selection strategy, the sibling class SimpleClientManager is employed. It returns the number of clients, registers them, samples the client, and returns the ClientProxies, based on the criteria specified for client selection.

4.3.5 Single-node Flower Simulation

In many situations, it may be of utmost importance to specify the required resources for a task. For instance, this could involve allocating CPUs and GPUs for computing in a machine-learning task. Ray [57] is a general-purpose, open-source cluster-computing framework that facilitates AI workloads such as simulation, training, rendering, and deployment. Ray will automatically detect the available GPUs and CPUs on the machine and allows specifying task resource requirements. The degree of concurrency between tasks scheduled by Ray is affected by their individual resource needs. In particular, a given machine's resources must be sufficient to support the currently executed tasks.

We employ the Flower single-node simulation concept for governing our federated environment. We utilize Flower's simulation function for this purpose. As shown below, the simulation function accepts as parameters the client function, client IDs, ray configurations, number of FL rounds, and Strategy. As previously explained, all user needs are synchronized by the Strategy. The FL loop then initiates the training and evaluation procedures.

```
fl.simulation.start_simulation(
        client_fn=client_fn,
        clients_ids = new_client_ids,
        client_resources={"num_cpus": 0.1},
        config=fl.server.ServerConfig(num_rounds=args.iterations),
        strategy=strategy,
        ray_init_args=ray_init_args,
    )
```

The VCE in Flower used to virtualize the clients in the system is built on Ray to schedule the execution of client tasks. When resources are inadequate, Ray can order the execution of client-side operations, allowing inexpensive computation of larger-scale experiments. Until the necessary resource requirements are satisfied, VCE holds off on creating such instances, which is advantageous in resource consumption computing.

The combination, as mentioned earlier, of Flower's VCE and Ray enables large-scale client simulation for evaluating system-level and algorithmic approaches. The proposed use cases employ this simulation to produce the fundamental outcomes in a large-scale automotive environment.

A UML diagram, as in Figure 4.13, illustrates the usage of the RayClientProxy and ClientProxy abstract classes in Flower.

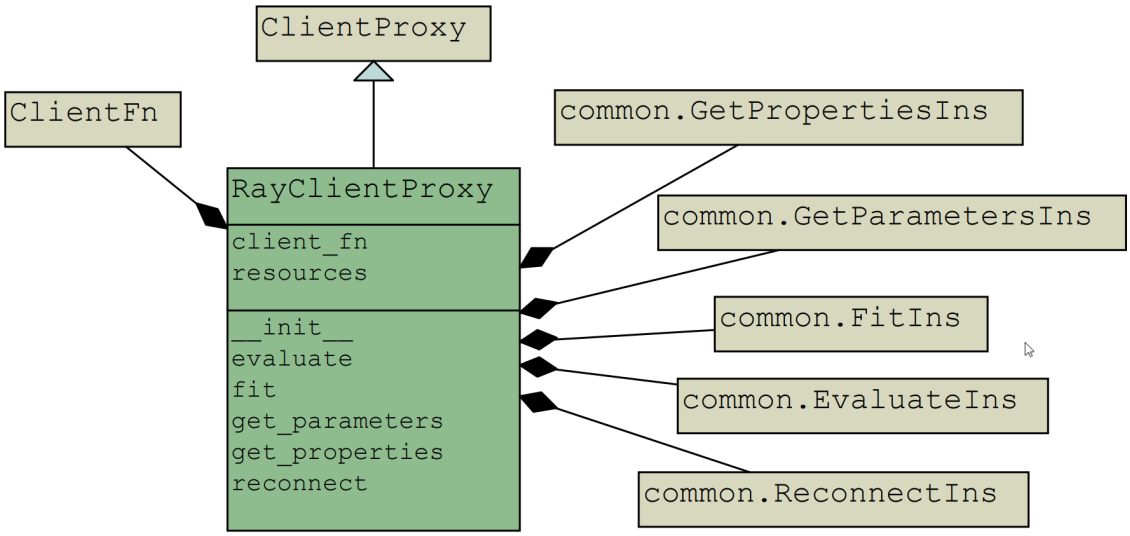

Figure 4.13: UML Diagram - RayClientProxy

4.3.6 End-to-end EV-DP Federated Model

We create the end-to-end differential private model for EV clients using the Opacus library. Opacus is an open-source, simple, adaptable, and fast Pytorch framework that provides differential privacy implementation for deep learning applications [98].

Following the Section 2.1.5' definition of DP, we monitor the privacy budget to determine the level of privacy attained by the EV clients. The implementation is straightforward using the PrivacyEngine class in Opacus. The user provides the noise multiplier σ (i.e., the amount of noise to be added to the model updates), l2-clipping norm (C), and delta value δ. Finally, each client's privacy budget (privacy loss ϵ) is calculated at the end of each FL round.

A sample code snippet of using the PrivacyEngine class is as shown below,

```
    PRIVACY_PARAMS = {
    "target_delta": ###,
    "noise_multiplier": ###,
    "max_grad_norm": ###
    }

privacy_engine = PrivacyEngine(
            model,
            sample_rate=sample_rate,
            target_delta=PRIVACY_PARAMS["target_delta"],
            max_grad_norm=PRIVACY_PARAMS["max_grad_norm"],
            noise_multiplier=PRIVACY_PARAMS["noise_multiplier"],
        )
```

As shown in Figure 4.14, the Opacus classes wrap the underlying object and carry out the following operations:

- Opacus computes the per-sample gradient and then clips the values to a user-specified l2-clipping norm
- The calibrated Gaussian noise the user provides is added to increase the randomness of model updates and preserve privacy.
- Later, we introduce Poisson sampling, wherein each sample from the dataset is included in each training phase with a certain probability p.

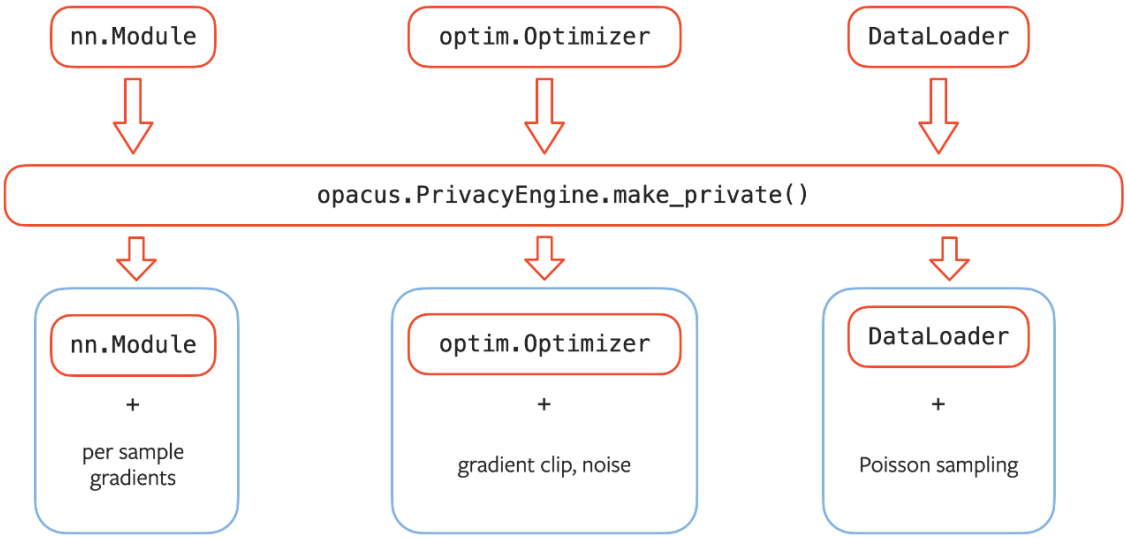

Figure 4.14: Main blocks provided by the Opacus library for DP [98]

4.4 Integrating W&B into the FL system

The difficulty of collaborating with multiple clients in a federated automotive environment is more complex than in a centralized environment. Using the W&B MLOps platform, it is possible to mitigate this issue, improve the experiment tracking, enhance interactive visualizations, record artifacts, and generate reports. Using W&B in the proposed configuration, scaling the process to thousands of clients, and collaborating with them are simple. The various outcomes for clients in the FL communication rounds through tracking are possible here. It is also possible to detect client errors and send them updates. In addition, it is crucial to understand system performance and track system resource usage, which is possible with W&B. Multiple combinations of the train and evaluation clients are stored and readily accessible, allowing for easy comparison with other experiments to determine which yields the best results.

As illustrated in Figure 4.15, we utilize the W&B blocks to develop a production-grade automotive setup for the use case described in the previous section, demonstrating the feasibility of using an MLOps platform in an FL environment for production-ready automobiles.

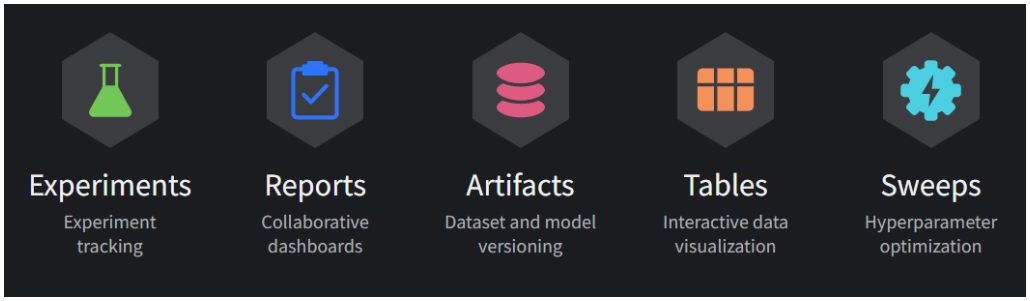

Figure 4.15: Integral components of W&B [6]

1. **Experiments**
   For experiment tracking, we must initiate a project, configure the parameters, and log metrics and other model artifacts, such as model weights. Following are some of the W&B APIs that we use for this purpose.

   The `wandb.init()` method is invoked at the start of our FL code to launch a new run. A new background process is spawned to log data to the newly created run. Additionally, a local directory is created to store all logs and files. The data is automatically synced to wandb.ai, allowing us to visualize the various metrics. New rounds are created as the FL round progresses, and the dashboard displays the corresponding results. We organize the run based on the FL round number using the `group` argument, which is especially useful for monitoring large simulation experiments. We then terminate the run and do any necessary cleanup by invoking the `wandb.finish()` method.

   A sample code snippet to illustrate the above explanation,

```
    run = wandb.init(project=proj_name, group="FL_SrvRnd_" + str(round_number), reinit=
True)
    wandb.run.name = 'FL_SrvRnd_' + str(config["round_number"])
```

2. **Tables**
   Using W&B's interactive Table, we can organize tabular data for regression metrics like MSE, RMSE, MAE, and Adjusted R2 in various ways, including grouping, sorting, filtering, generating calculated columns, and making charts. We can easily select the relevant clients and view their log data tables. A sample code snippet to illustrate the above using the `wandb.log()`, method is as shown below,

```
    wandb.log({"Avg test loss" : wandb.plot.line(table, "Client ID", "Avg test loss",
title="Avg Test Loss for Evaluation Clients")})
```

3. **Reports**
   W&B facilitates sharing graphs, visualizations of how model versions have improved, notes, and dynamic experiments in various flexible file formats. It is capable of collaborating with other members of the team on time. We can keep track of the results and plan the next steps by storing the work logs. Other features include real-time comments, snapshots of work logs, and customizable reporting as a LaTeX zip file or PDF.

4. **Artifacts**
   We capture every FL round and compare model versions to identify the best results for production. The artifacts registry holds all the necessary information here.

4.5 Employed Evaluation Metrics

Prediction of a time series or regression problem requires mapping input variables to continuous output variables(s) [64, 67]. In such situations, various metrics are employed to evaluate the obtained results. This section focuses on the essential metrics for the proposed work. We explain the use metrics for Use-case (a) as described in the Section 4.1.1 to describe the mathematical notations of the metrics for estimating the battery capacity of EVs. These metrics are also used for evaluating the results obtained for the Use-case (b) (Section 4.1.2)

Mean Absolute Error (MAE)

The MAE is defined as the average of the absolute difference between forecasted and actual values. The actual battery capacity at an instant of time i is $y_{(\text{battery\_capacity})\text{i}}$, and the predicted value at the same instant of time i is $\widehat{y_{(\text{battery\_capacity})\text{i}}}$. The error term here is $y_{(\text{battery\_capacity})\text{i}}$ - $\widehat{y_{(\text{battery\_capacity})\text{i}}}$. The error term's magnitude can be positive or negative depending on the predicted value. However, for calculating the MAE, we take the average of the absolute difference between the actual and predicted values. The total number of samples here is N. This is as explained in the equation below,

$$MAE = 1/N \sum_{i=1}^{N} |y_{(battery\_capacity)i} - \widehat{y_{(battery\_capacity)i}}|$$

The smaller the MAE, the more closely a model fits a dataset. Consequently, the MAE will indicate the average forecast error we can anticipate. It is reasonably resistant to outliers. However, it can be an inadequate measure when the data points to extreme values.

Mean Squared Error (MSE)

MSE is a prevalent and widely utilized metric. MSE facilitates the estimation of the predictive or forecasting model's quality. MSE is always a positive value, and a value closer to 0 or a lower value indicates high performance. Similar to the MAE, we consider the square of the error term here. MSE is a more decisive metric because it considers the difference between actual and predicted values, known as variance and the bias, a shift in the distance of predicted value from its actual value. In addition, as a result of the square error term, it penalizes more significant errors or outliers. MSE is advantageous when prediction values are widely dispersed, and larger values must be penalized.

$$MSE = 1/N \sum_{i=1}^{N} (y_{(battery\_capacity)i} - \widehat{y_{(battery\_capacity)i}})^2$$

Root Mean Squared Error (RMSE)

RMSE can be directly interpreted in terms of measurement units. The RMSE is calculated by taking the square root of the MSE. It has the same units as the vertical or Y-axis quantity; here, the instantaneous battery capacity is measured in Wh.

$$RMSE = \sqrt{1/N \sum_{i=1}^{N} (y_{(battery\_capacity)i} - y_{(\widehat{battery\_capacity})i})^2}$$

Adjusted R-squared or Adjusted R2

The R-squared (R2) value, also known as the coefficient of determination, is a statistical measure of the performance of a regression model. In a regression model, R-squared describes the proportion of variance for a dependent variable (y) concerning an independent variable (x) or variables (x1, x2, ., xn). The value of R-squared is always between 0 and 1. As the value tends toward 1, it indicates an accurate prediction.

The difference between the R-squared value and the Adjusted R-squared value is that the R-squared value assumes all independent variables influence the model's outcome. In contrast, the Adjusted R-squared value takes into account only the independent variables that significantly affect the model's performance. This means that R-squared will always increase when a new variable is added to the prediction model, but the Adjusted R-squared will decrease if the variable is inconsequential.

As shown in the equations below, we use the Adjusted R-squared to determine the model accuracy in our experiments. Here k is the number of independent variables (i.e., number of attributes in the feature space).

$$R^2 = 1 - \frac{\sum_i^N (y_{(battery\_capacity)i} - y_{(\widehat{battery\_capacity})i})^2}{\sum_i^N (y_{(battery\_capacity)i} - \overline{y})^2}$$

$$Adjusted\_R^2 = 1 - [\frac{(N-1)}{N-k-1} * (1 - R^2)]$$

5 Experimental Setup

The SUMO simulator for data acquisition and the elements necessary to render the EV and CS data, as was explained in the preceding "Methodology" chapter. Then, we discussed a comprehensive overview of the Flower federated framework and its fundamental primitives and design principles.

This chapter dives deeper into the experimental setup for the proposed system using the design concepts outlined in the previous chapter,

5.1 **Scenario Generation in SUMO**
Here, we discuss the nuance attributes utilized during the SUMO simulation process. Following this, we discuss the various input and output features of the EVs and CSs as a result of the simulation process.

5.2 **Data Distribution - Clients' Heterogeneity**
For the obtained simulation data, we discuss data heterogeneity (non-IID) for the EVs and CSs, one of the critical governing factors of FL. Further, we provide a mathematical explanation to recognize the generated data's diversity and non-uniformity.

5.3 **Local Baseline (Personalized) Model**
We discuss the modeling of the local (personalized) model and its pertinent hyper-parameters for each client (EVs and CSs).

5.4 **System Setup**
We discuss a detailed explanation of system configuration for the two use cases described in the previous chapter. We comprehend the interaction of SUMO simulator data with the Flower framework to establish an FL environment in detail.

5.5 **FL Training**
Finally, we describe the resources necessary for FL training and the training procedure.

5.1 Scenario Generation in SUMO

Using OpenStreetMap, we generate a 417-square-kilometer map of Stuttgart's location. The map obtained is in .osm format. The netconvert command line application converts the map_STR.osm file to a map_STR.net.XML file that the simulator can interpret. Many parameters used by netconvert determine how the network is imported and how the SUMO network is generated. For this purpose, we employ the following processing methods,

- –geometry.remove : Swaps out nodes that define only edges with geometry points
- –roundabouts.guess : Activates round about guessing

- –ramps.guess : Most imported network descriptions lack information regarding highway on and off ramps, indicating that ramps connect to the highway without acceleration/deceleration lanes. By enabling this setting, we can make educated guesses about where on and off ramps should be constructed.
- –junctions.join : To connect nearby junctions we use this option
- –tls.guess-signals : Enables traffic lights guessing
- –tls.discard-simple : It does not activate traffic signals on geometry nodes loaded in a format other than plain XML
- –tls.join : Attempts to group nodes that are under traffic lights (tls) control

The commands listed below correspond to each of the steps discussed later. We generate the necessary files using these commands.

```
$ netconvert --osm-files map_STR.osm --output-file map_STR.net.xml --geometry.remove --
roundabouts.guess --ramps.guess --junctions.join --tls.guess-signals --tls.discard-simple --
tls.join

$ python utils\randomTrips.py -n map_STR.net.xml --fcd-output -o map_STR.trips.xml -r
map_STR.rou.xml -fcd-output.vType -e 101000 --period 100 --additional-file vehicles.add.xml --
fringe-factor 1 --trip-attributes="type=\"typedist1\"" --verbose
```

We use the randomTrips.py utility from SUMO to generate the random trips for the EVs for a given network (option -n). The resulting generated random trips are stored in the map_STR.trips.XML file. The map_STR.rou.XML file generates the routes taken by EVs that are compatible with duarouter (option -r). The trips are evenly distributed within the interval specified by begin (option -b, default 0) and end (option -e, 101000) in seconds. The repetition rate determines the number of trips in seconds (option -p, 100) for each of the EVs. The total number of vehicles in the simulation is determined by the values of start (defaults to 0), end (101000), and period (100), adhering to the formula specified in Section 4.2.2. Every vehicle has a unique vehicle id (vID) and vehicle type (vType) as specified in all the generated XML files.

Table 5.1: Maximum speed, acceleration, and battery capacity for 7 distinct EVs

vType	speed ms^{-1}	acc. ms^{-2}	battery cap. (Wh)
soulEV65	29.06	1.0	64000
VW_eUp	54.44	17.8	32300
VW_ID4	50.00	8.5	77000
BMW i3	41.67	7.3	39000
Kia EV6 GT	72.22	3.5	77400
SUV	60.00	20.0	80000
VW_ID3	44.44	7.9	58000

Table 5.2: User-defined values for speedFactor and speedDev for 7 distinct EVs

vType	speedFactor	speedDev
soulEV65	0.2	0.2
VW_eUp	1.3	0.8
VW_ID4	1.8	1.5
BMW i3	0.3	0.5
Kia EV6 GT	2.0	1.5
SUV	2.0	1.5
VW_ID3	1.5	0.8

Maximum speed, acceleration, and battery capacity for seven different EVs are illustrated in the Table 5.1. We create an additional file (Listing 5.1) containing the seven unique vehicle attributes of the BMW i3, SUV, Volkswagen ID.3, Volkswagen ID.4, Volkswagen e-up, Kia EV6 GT, and Soul EV65 using the details in the Table 5.1. We can use this additional file in conjunction with the —trip-attributes option to select vehicles with specific characteristics. Setting the —fringe factor to 1 increases the likelihood of trips beginning and ending at the network's perimeter. We randomly chose the values for speedFactor and speedDev to adjust and exceed the EV's speed limits in a particular lane (Table 5.2). Setting the speed deviations is primarily intended to prevent vehicles from exhibiting identical behavior, thereby facilitating the achievement of a non-IID data set. Certain specific EVs lack a defined route. Consequently, such EVs are removed from the route file.

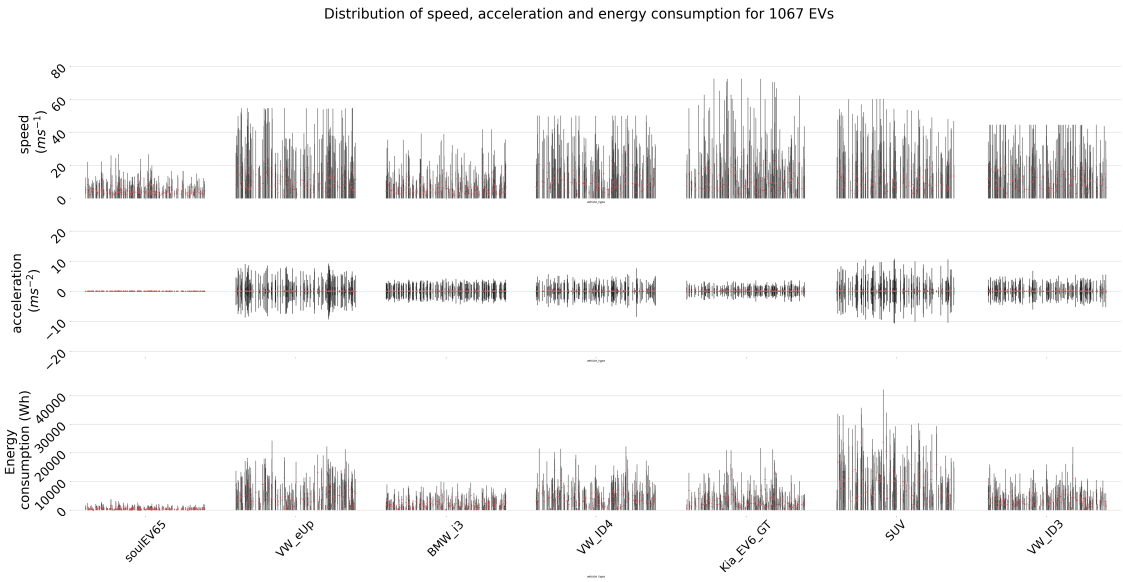

Figure 5.1: Speed, acceleration and energy consumption boxplot distribution for 1067 EVs belonging to the 7 distinct EV categories

We finally pass the generated map_STR.net.xml and map_STR.rou.xml files to the sumo configuration file. The simulation can then be initiated in the simulator using the configuration file. After successful simulation, the final output is an XML file containing, among other essential headers, the vehicle ID (vID), vehicle type (vType), routes taken by the vehicle, speed, acceleration, energy consumed, and battery capacity.

For the simulation setup mentioned above, we simulate the behavior of around 1067 EVs, each with its unique vehicle attributes. The Figure 5.1 illustrates the resulting variations in speed, acceleration, and corresponding energy consumption for 1067 EVs.

Charging Stations

We established five remote CSs in Esslingen, Ludwigsburg, Mitte, Renningen, and Sindelfingen, all within the city of Stuttgart. Figure 5.2 depicts the number of EVs utilizing these installed CSs. We utilized netedit as explained in Section 4.2.2, accessed the map STR.net.XML file, and installed CSs

Listing 5.1 Sample format of the Additional XML file containing EVs' attributes

```
    <?xml version="1.0" encoding="UTF-8"?>
    <additional>
     <vTypeDistribution id="typedist1" >
        <vType actionStepLength="1.0" color="blue" emissionClass="MMPEVEM" id="BMW_i3" vClass
="passenger" maxSpeed="41.67" accel="7.3" decel="7.3" speedFactor="0.3" speedDev="0.5">
        <param key="has.battery.device" value="true"/>
        .....................
        </vType>
        <vType actionStepLength="1.0" color="blue" emissionClass="MMPEVEM" id="SUV" vClass="
passenger" maxSpeed="60" accel="20" decel="20" speedFactor="2.0" speedDev="1.5">
    <param key="has.battery.device" value="true"/>
        .....................
        </vType>
        ..........................................
     </vTypeDistribution>
</additional>
```

Listing 5.2 Sample format of the Additional XML file containing installed CSs' information

```
<?xml version="1.0" encoding="UTF-8"?>
<additional xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/additional_file.xsd">
<!-- StoppingPlaces -->
    <chargingStation id="cs_0" lane="24833659_0" startPos="23.68" endPos="33.68" friendlyPos="
true" power="22000.00" chargeInTransit="1"/>
    ...............
    <chargingStation id="cs_10" lane="-365894818#3_0" startPos="25.76" endPos="35.76"
friendlyPos="true" power="22000.00" chargeInTransit="1"/>
</additional>
```

on randomly selected eligible lanes. We then export an additional XML file (Listing 5.2) containing the details of these installed CSs from the netedit tool. We begin the simulation by implementing the EV simulation procedure but with the positioned CSs. EVs will then utilize the CSs to charge when necessary.

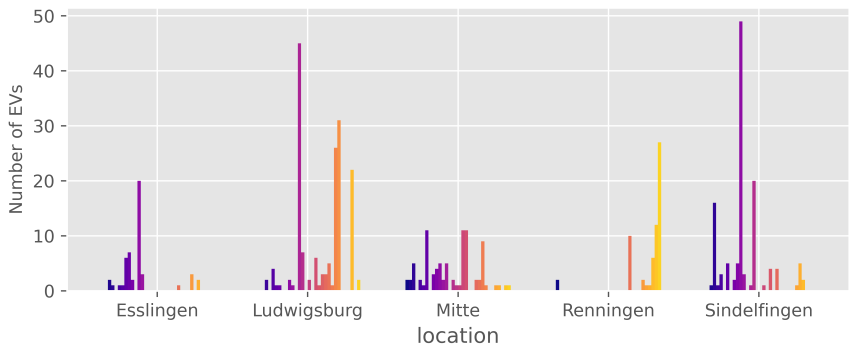

Figure 5.2: CSs deployed at different locations and number of vehicles using them

Simulation Output

Multiple parameters regarding the EV are generated as a result of the simulation output. We select the four necessary headers for comprehending the impact of speed, acceleration, and energy consumption on battery capacity degradation. Figure 5.3 depicts, for vehicle ID 0, an example of these four variables (velocity, acceleration, energy consumption, and battery capacity decline) when the EV is in motion. As the vehicle moves, the vehicle accelerates and speeds up at specific positions on the lane, and its energy consumption increases; we observe a gradual decrease in battery capacity. Listing 5.3 provides a preview of a battery output file reported in XML.

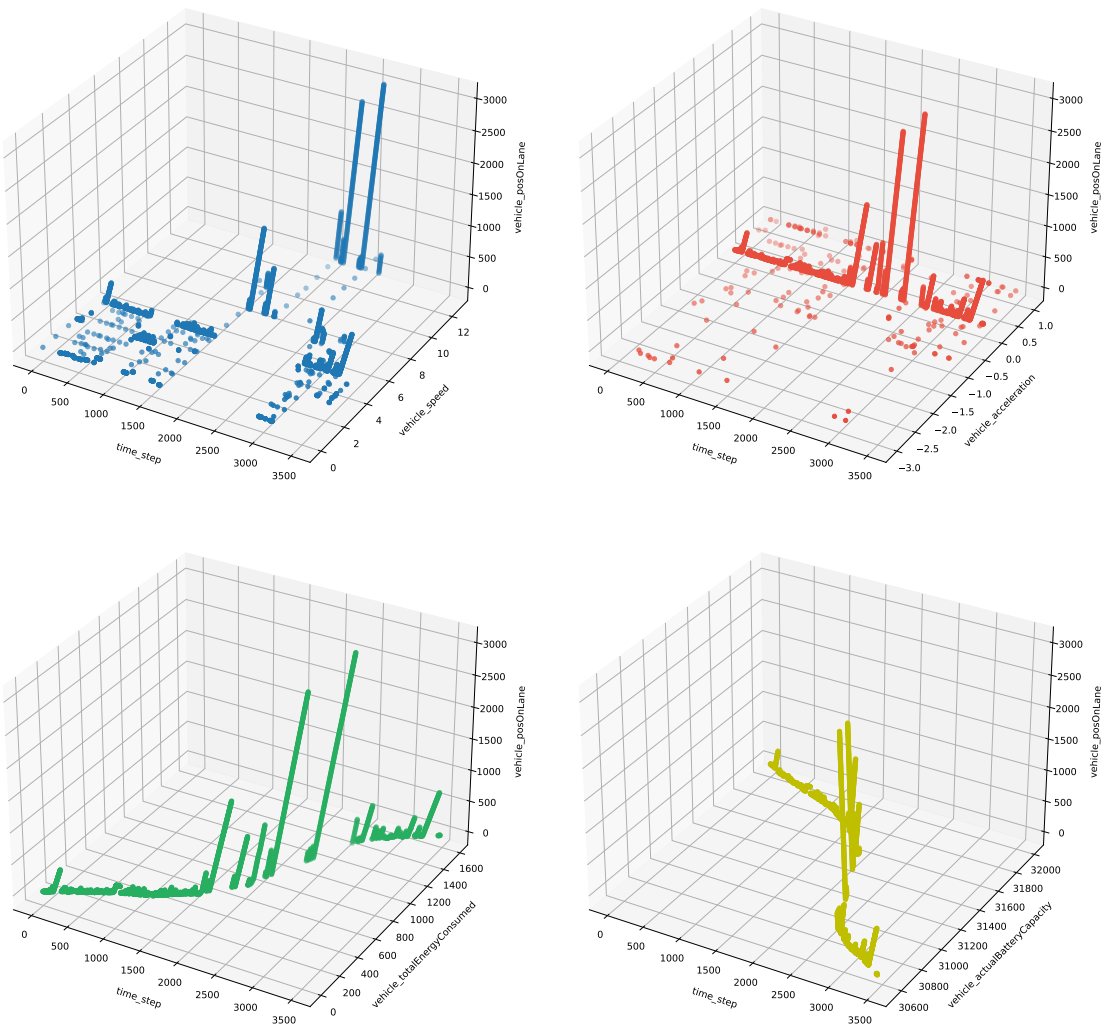

Figure 5.3: Variation of speed, acceleration, energy consumption, and battery capacity for the EV vID 0

Listing 5.3 Sample format of the battery output XML file

```
<?xml version="1.0" encoding="UTF-8"?>
    <battery-export>
        <timestep time="0.00">
            <vehicle id="0" energyConsumed="0.0000" totalEnergyConsumed="0.0000"
totalEnergyRegenerated="0.0000" actualBatteryCapacity="32000.0000" maximumBatteryCapacity
="64000.0000" chargingStationId="NULL" energyCharged="0.0000" energyChargedInTransit="0.0000"
energyChargedStopped="0.0000" speed="0.0000" acceleration="0.0000" x="12532.2604" y
="4758.6699" lane="23435862#15_0" posOnLane="5.1000" timeStopped="0"/>
        </timestep>
    ......................................................................
         <timestep time="20746.00">
            <vehicle id="164" energyConsumed="0.4277" totalEnergyConsumed="1116.2333"
totalEnergyRegenerated="123.8385" actualBatteryCapacity="31007.6052" maximumBatteryCapacity
="64000.0000" chargingStationId="NULL" energyCharged="0.0000" energyChargedInTransit="0.0000"
energyChargedStopped="0.0000" speed="3.9670" acceleration="0.0000" x="14200.4906" y
="1295.4643" lane="23424772_0" posOnLane="8.7370" timeStopped="0"/>
    ......................................................................
        </timestep>
    </battery-export>
```

We obtain the output of the CSs' energy consumption in an XML file as in Listing 5.4. We select the three necessary headers, 'chargingStation_id', 'step_time', 'step_partialCharge' for determining the energy-consumption from each utilized CSs. Figure 5.4 depicts an instantaneous, gradual increase in the energy consumption in Wh ('step_partialCharge') of the installed CSs when an EV is charged at a specific station.

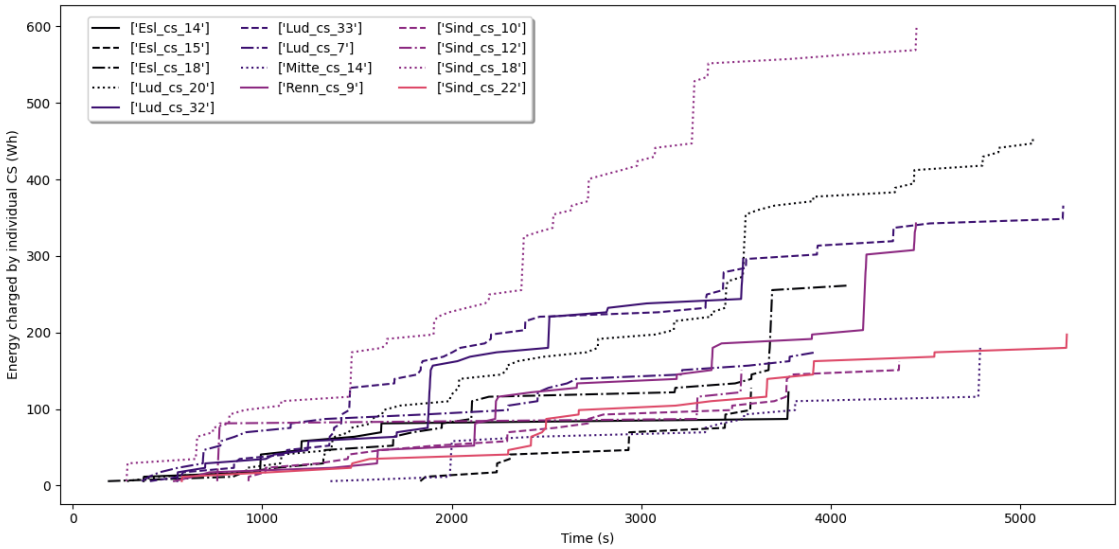

Figure 5.4: Illustration of energy consumption by each of the mentioned CS.

Listing 5.4 Sample format of the CS output XML file

```
<?xml version="1.0" encoding="UTF-8"?>
    <chargingstations-export>
        <chargingStation id="cs_27" totalEnergyCharged="34.83" chargingSteps="6">
        <vehicle id="63" type="BMW_i3" totalEnergyChargedIntoVehicle="34.83" chargingBegin
="2017.00" chargingEnd="2022.00">
            <step time="2017.00" chargingStatus="chargingInTransit" energyCharged="5.81"
partialCharge="5.81" power="22000.00" efficiency="0.95" actualBatteryCapacity="19200.24"
maximumBatteryCapacity="39000.00"/>
        ..............................................................
        </vehicle>
    </chargingStation>
</chargingstations-export>
```

5.2 Data Distribution - Clients' Heterogeneity

Clients participating in an FL environment may have varying data distributions, typical as in real-world use cases not meeting the IID nature of data. The amount of data distributed on each client node is always imbalanced in numerous applications. The varying feature space, labels, and attributes, among other factors with the various clients, can influence the skewness of the data distribution leading to the discussion of the non-iid data distribution. The non-IID ness of the data could lead to model divergence during FL training. When configuring an FL environment, it is crucial to be wary of the data's intrinsic characteristics and how to deal with variations in them to avoid model divergence. One of the most common issues with FL systems is dealing with non-IID data. Specifically, this occurs frequently in supervised learning settings. We provide justifications below for the non-IID data generated from the simulator and used in the current use cases defined [12].

1. EVs participating in a federated environment contain data samples collected individually under particular conditions leading to the generation of statistically unequal datasets. Every individual client proposes a unique behavioral imprint, directing the concept of data heterogeneity or non-IID data. In the current use-case of EV mobility and battery consumption, each client is represented as

   (5.1) $vID_{\mathrm{i}} \quad ; \quad i = \{1, 2, ..., n\}$

   where n being the total number of EV clients.

   The data space for this client is represented as

   (5.2) $data_{\mathrm{t}} = ((v, a, ec), \quad y) \in X \times Y$

   where (v, a, ec) represents the vehicle speed, acceleration and energy consumption at the given instant of time t and y is the corresponding battery consumption at that time.

   The join probability distribution for each client is given by,

   (5.3) $P((v, a, ec), y) = P((v, a, ec))P(y|(v, a, ec))$

The current equation upholds in a supervised learning environment where the features have corresponding labels.

Consider the data probability of the EV client vID_i, $P_{vID_i}((v, a, ec), y)$, and EV client vID_j $P_{vID_j}((v, a, ec), y)$. The unequal data probabilities between the two clients will result in the following,

$$P_{vID_i}(y|(v, a, ec)) \neq P_{vID_j}(y|(v, a, ec)) \tag{5.4}$$

The above mathematical demonstrations explain how EV owners in the current simulation setup have varying speed and acceleration thresholds, which alters the driving behavior pattern. We see the same in Figure 5.5. Client 1 with vID 82 and Client 2 with vID 83 exhibit different speed and acceleration patterns.

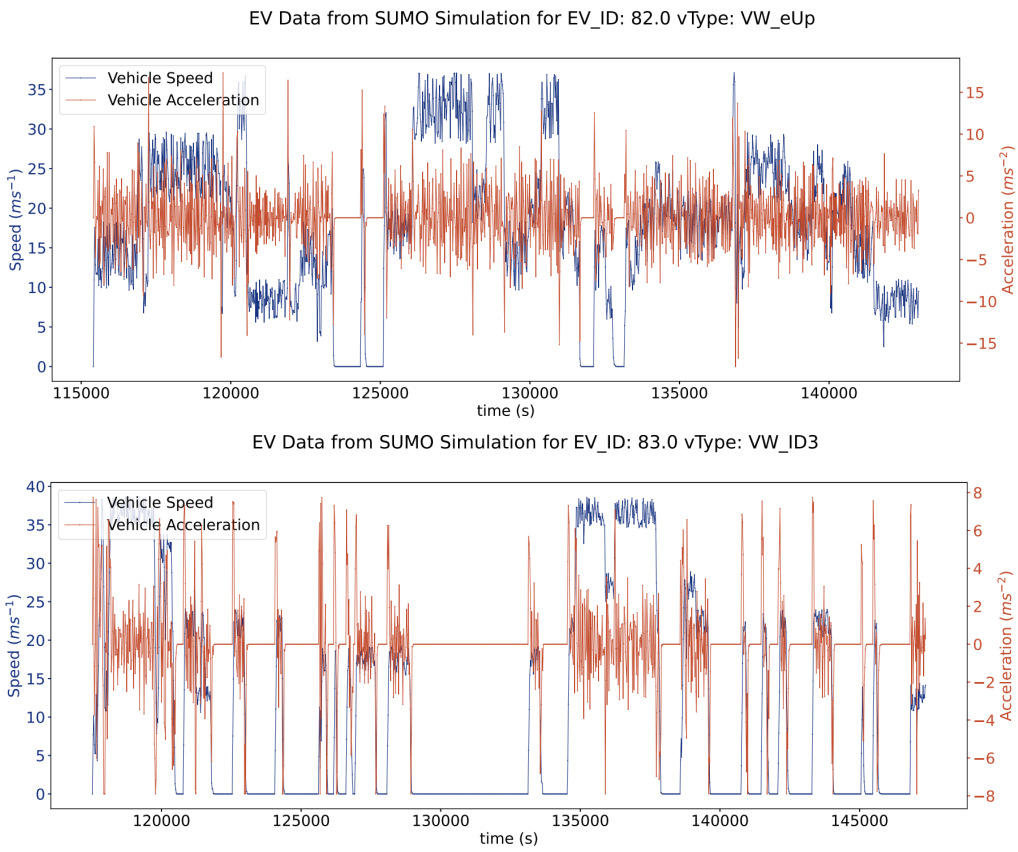

Figure 5.5: Varying speed and acceleration patterns for two different clients

2. For the battery capacity data of the EVs and the energy demand of the CSs here, every client experiences change over time. We ascribe this to temporal skewness [103] in the time-series data distribution, contributing to data heterogeneity. In the context of time series data, the distribution is the dynamic time index associated with each client. It is a common type of skewness in data sets that record data with timestamps observed on many devices. In equation 5.5, we see that the battery capacity declines as the EV is in motion resulting in instantaneous change. t is the time index here.

$$P_{vID_i}((v, a, ec), \quad y|t) \tag{5.5}$$

The energy consumption for any given CSs, at an instant of time t, is given by ec(t). In distributed time-series data, we consider four lag observations (explained in the upcoming section) to predict the energy consumption at time t. As given in equation 5.6, the data distribution changes over time for each CS client indicating temporal skewness. Figure 5.6 illustrates the above explanation of temporal skewness for the selected EVs and CSs.

$$P_{\mathrm{vID_i}}((ec_{\mathrm{val}}(t-4), ec_{\mathrm{val}}(t-3), ec_{\mathrm{val}}(t-2), ec_{\mathrm{val}}(t-1)), \quad ec_{\mathrm{val}}(t)|t) \tag{5.6}$$

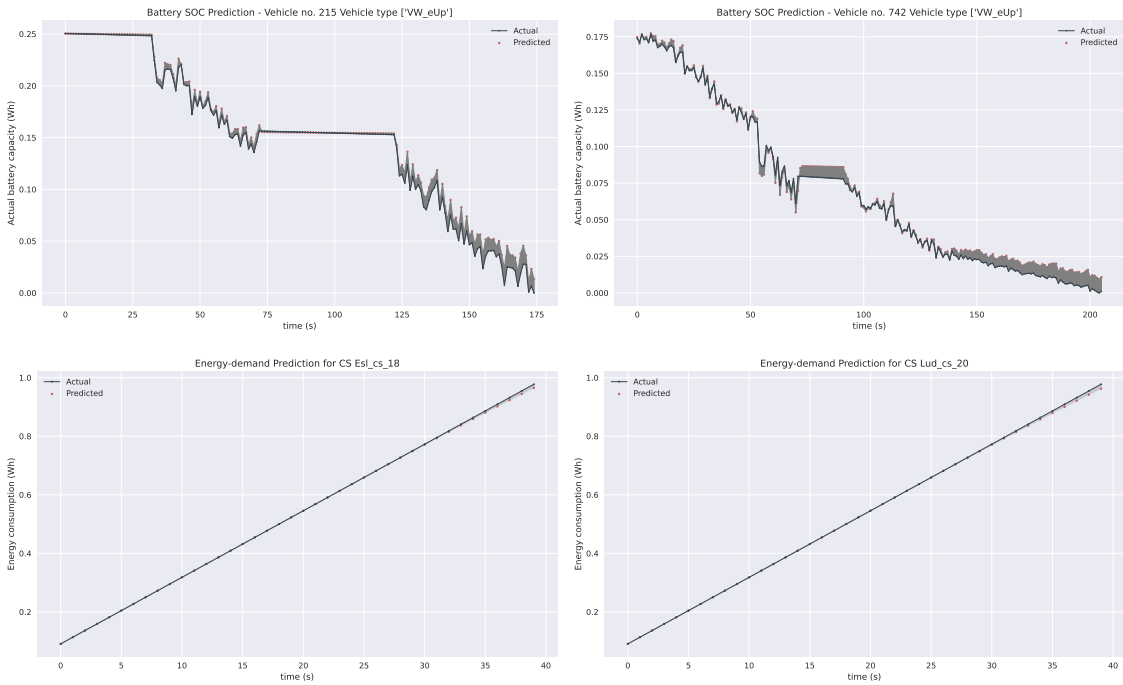

Figure 5.6: Temporal skewness for EVs and CSs

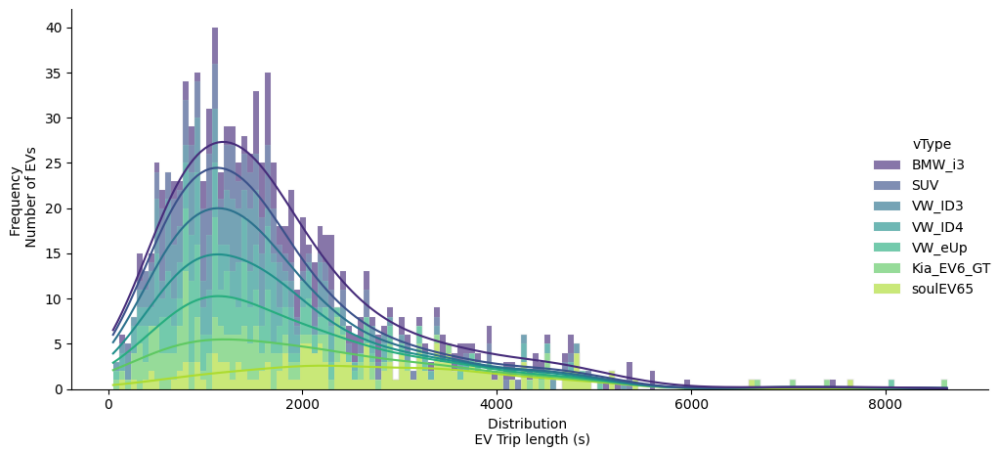

Figure 5.7: Kernel density estimate for the distribution of observations in the trip length for 7 different EVs.

3. Figure 5.7 depicts the continuous probability distribution for the long and short trips taken by 1067 EV owners. EVs' data here exhibit a combination of long and short trips. In due course, the commutes made by EVs may change. This uneven data distribution caused by the varying length of the data samples leads to the non-IID distribution of the data. We refer to this characteristic as quantity skew [103].

5.3 Local Baseline (Personalized) Model

In situations involving sequential data from instantaneous, nonlinear systems, temporal dependencies between past and future values are of utmost importance. As discussed in the Section 2.2, LSTM networks are one type of RNN that specializes in learning such temporal dependencies. LSTMs effectively map the input sequences of EVs to predict battery capacity. Similarly, they are also used to forecast energy demand for CSs.

- **EV local model**
  We have adopted the LSTM-Pytorch [63] framework across all of our local EV clients. The LSTM layer used for the battery lifetime prediction consists of three input features: vehicle speed, acceleration, and energy consumed. The number of features in the hidden layer is 32. We have a fully connected linear output layer that predicts the actual battery capacity. We use the hyperbolic tangent function as the final activation layer to obtain the output. As the model training begins, we use back-propagation to fine-tune the network by computing the gradient of the loss function. We then update the weights and minimize the loss. We use the MSE and Adam as the loss function and optimizer, respectively. The learning rate for each client is 0.01, and we train for 2000 local epochs.

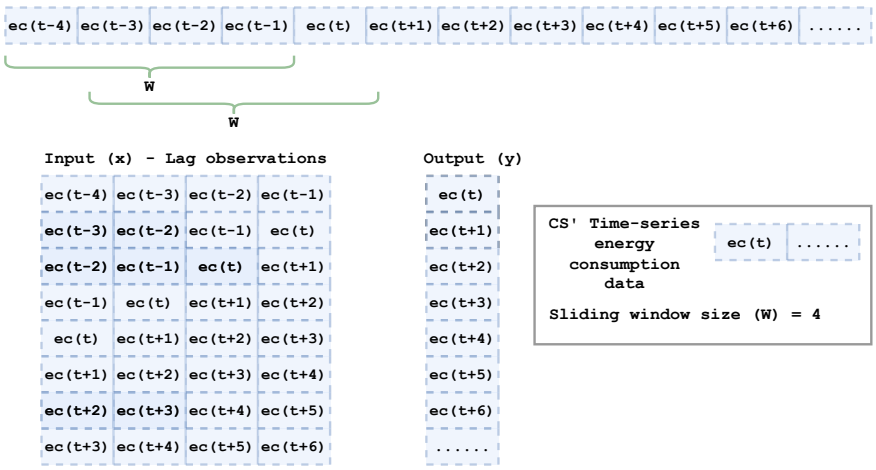

Figure 5.8: CS' time-series energy consumption data to a supervised dataset

- **CS local model**
  We use another LSTM-Pytorch network to obtain the prediction values for the energy consumption forecasting from each CS. Four lag observations (input (X)), are used when we transform the time-series data into supervised learning, ec_val(t-4), ec_val(t-3), ec_val(t-2), ec_val(t-1), to predict the forecast value (output (y)) of ec_val(t). Using this method, we

convert the time-series energy consumption of each CS to a supervised dataset, as illustrated in Figure 5.8, ensuring temporal dependency. Further, we use a single-layer LSTM with 32 hidden cells. Following is the linear layer that outputs the forecast value.

Figure 5.9 illustrates the above models, EV (left) and CS (right) local models generated using the Netron visualizer [72].

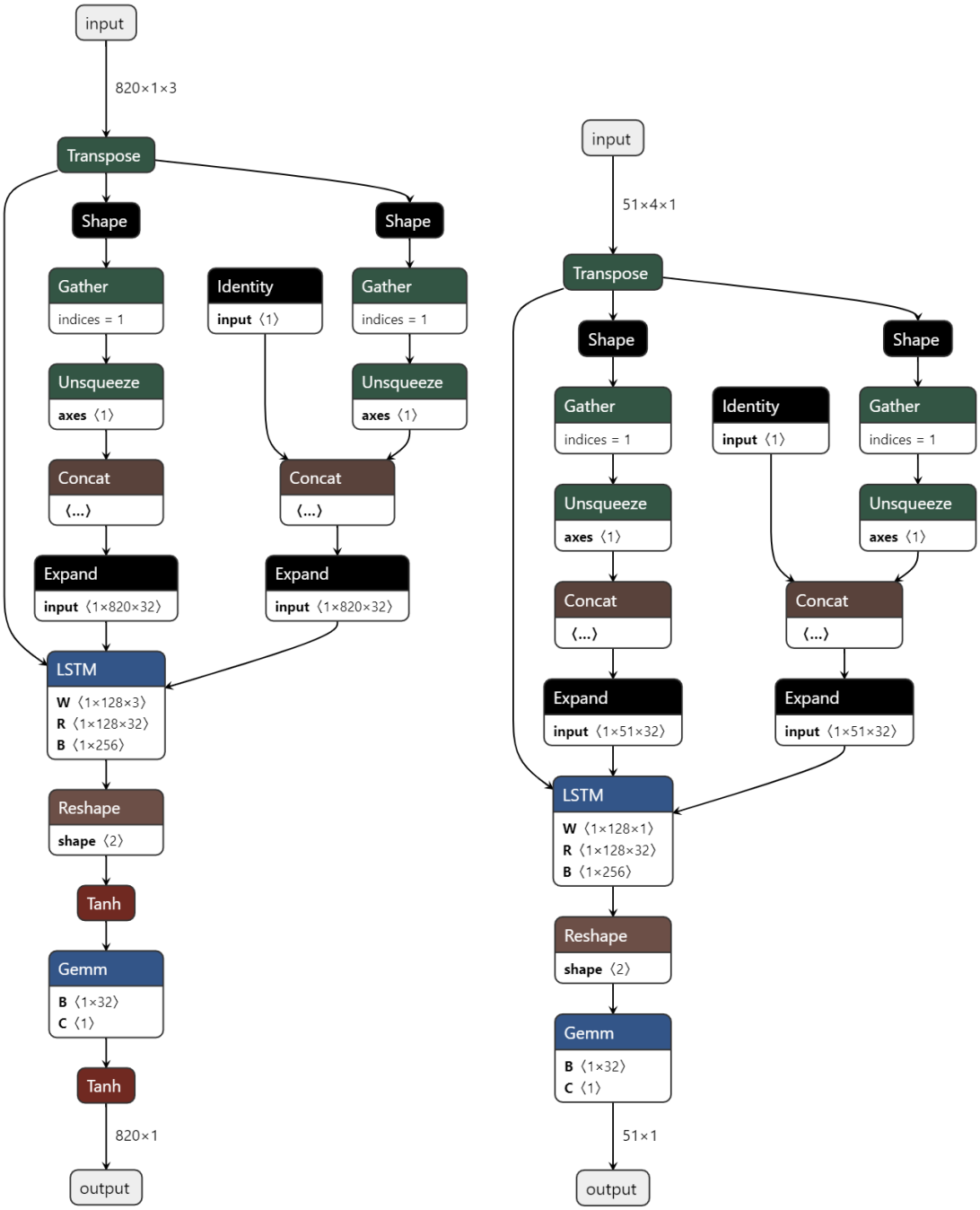

Figure 5.9: Layers of the LSTM-Pytorch model for EV (left) and CS (right)

5.4 System Set-up

This section explains the bringing together of the use cases, simulation data, and the Flower framework. The illustrations below help us understand how the overall process is interwoven to set up a federated environment.

As depicted in Figure 5.10, we combined the simulation data for 1067 EVs using the Flower framework to create a federated environment. As previously explained, the user selects clients for training and evaluation. The ClientManager efficiently manages the registration and sampling of these particular clients.

Here, as depicted in the example diagram, client ID vID_2, vID_86, and vID_1008 are chosen for the training (fit()) round and ClientProxies are generated for these clients. The inactive clients are also shown in the figure. After the initial round of FL, the server receives each client's model parameters and weights. The server implements the FedAvg algorithm and shares a global model for evaluation. The clients chosen for evaluation here are vID_15 and vID_3. They obtain global model updates and initiate the evaluation process. The client utilizes this model to test on their unseen test data. In addition, the regression metrics are used to determine the error rate and accuracy of prediction, indicating model performance. Typically, the first round of the FL yield divergent results. Therefore, we conduct two additional FL rounds to achieve the required convergence. It is to be noted that we assume that no client is faulty for this use case.

We distributed the CSs throughout various geographical regions. During mobility, the EVs charge at the installed charging stations. As illustrated in Figure 5.11, we choose CSs CSID_2, CSIS_86, and CSID_n are chosen for the training process. CSID_n fails to participate in training owing to inadequate data samples. Consequently, it returns empty parameters to the server. Upon encountering such a circumstance, the server notifies the CSID_n that it has not been selected for training and labels it a defective client. The server then computes the aggregation only from CSID_2 and CSID_86. The FaultTolerantFedAvg algorithm handles the fault tolerance mechanism by not halting the complete FL process because of the faulty clients. Further, the global model updates are sent to CSID_15 and CSID_3. These clients further evaluate the energy demand for them. The FL training is conducted for another two rounds.

5.5 FL Training

We are using a single machine to host one server and numerous EV clients (here, we use at most 1000 clients). Following the execution of the server, the client process begins. The standalone machine used here for the single-node Flower simulation consists of 64 CPU cores and 256 GB of RAM. The Ray server manages the resource consumption by utilizing the CPUs in a resource-efficient manner. Furthermore, we can configure the utilization of the CPU cores during training using the ray arguments "num_cpusïn the start_simulation() of the Ray-based Flower server.

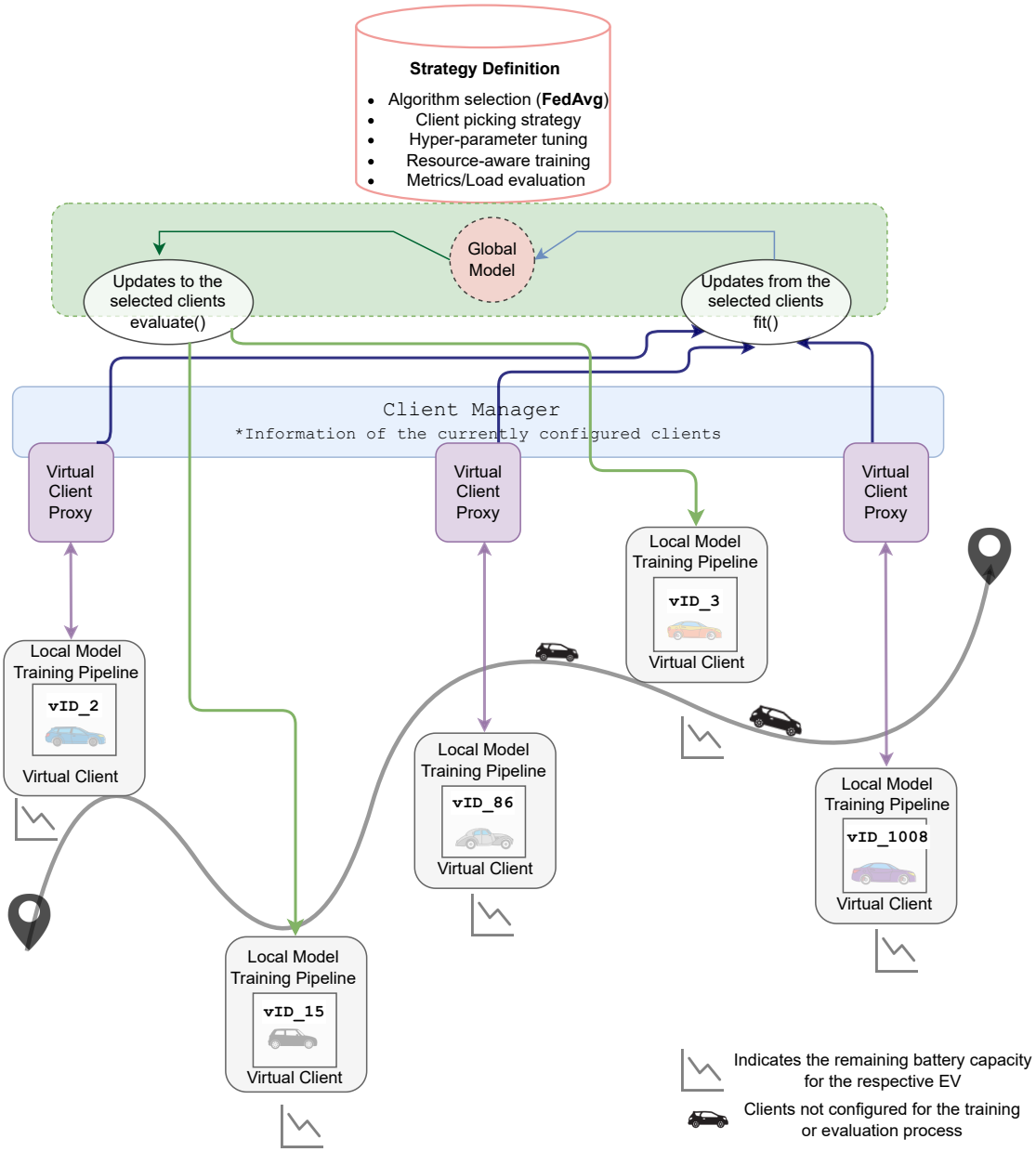

Figure 5.10: System Design - Battery capacity prediction for the EVs

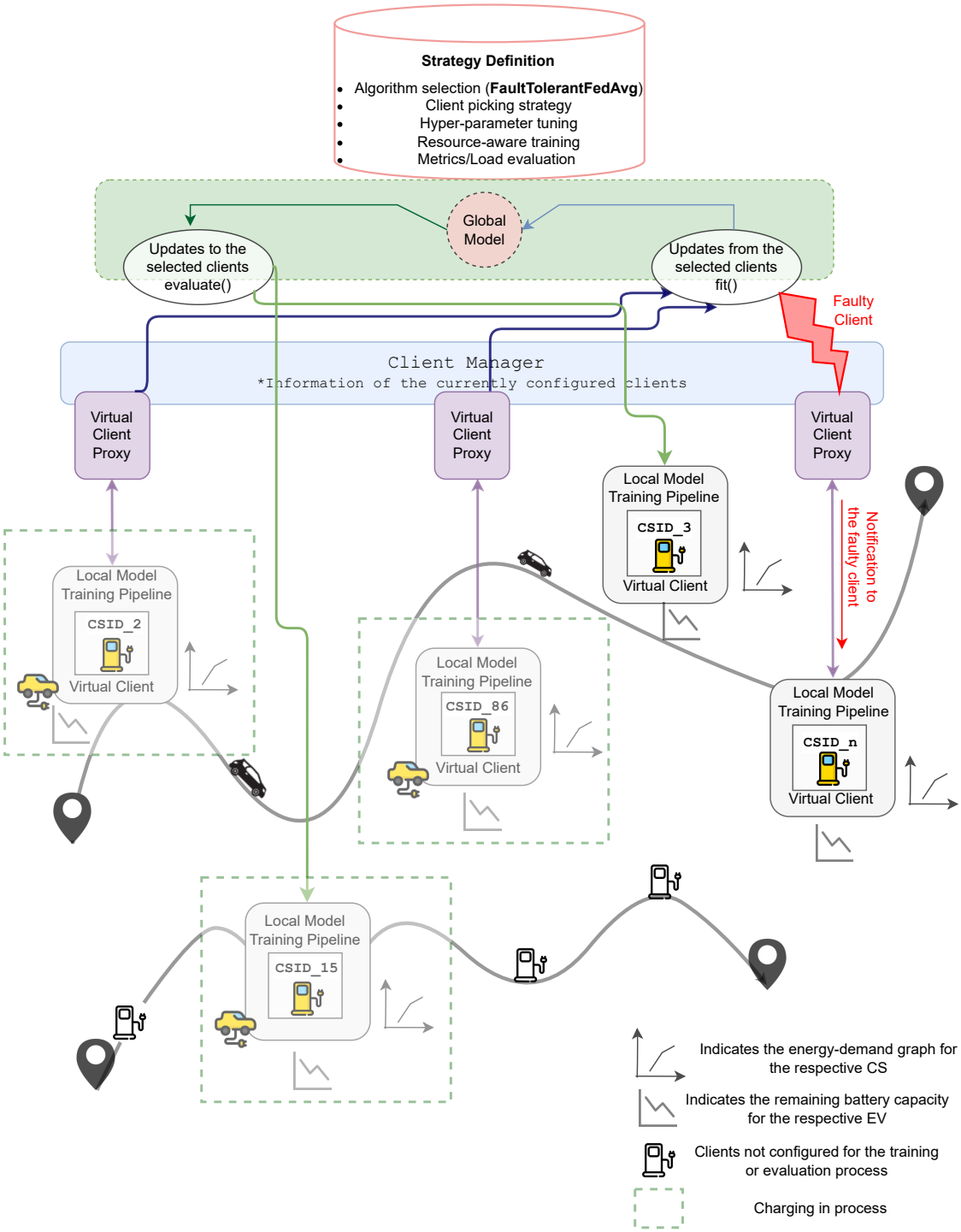

Figure 5.11: System Design - Energy-demand forecasting for the CSs

6 Results and Observations

This chapter will detail the results of our FL environment setup implementation. We discuss the results for the implemented Use-case (a): Actual battery capacity prediction for EVs and Use-case (b): Energy-demand forecasting for CSs explained in Section 4.1.1 and Section 4.1.2, respectively. Furthermore, this chapter addresses the three main research questions described in Section 1.2.

We comprehend how Flower's single-node simulation can adapt to different user-specified strategies, such as active client contribution and fault tolerance during deployment, and accommodate production-level algorithmic and research requirements. We discuss how the implemented FL system in Figure 5.10 and Section 5.11 intends to protect user privacy by providing converging results over various FL iterations for EVs and CSs, respectively. We also compare and contrast FL with the local baseline model (personalized model) and discuss the former's advantages over the latter. Using W&B, we provide an interface for the experimental tracking and report artifacts of different results across multiple rounds in FL. Finally, we discuss the privacy guarantee results of the end-to-end DP model applied to the implemented FL system for each local client. In addition, we shed light on how DP-related hyper-parameters influence overall system performance and the trade-offs that must be considered when designing a DP-FL system. In the concluding section of the chapter, the obtained results are discussed.

6.1 Performance Results for Use-case (a): Actual battery capacity prediction for EVs

6.1.1 FedAvg Algorithm's Performance Evaluation

The experimental settings here involve training and evaluating different combinational splits of clients. We use the FedAvg algorithm and calculate the MSE, RMSE, MAE, and Adjusted R2 scores across multiple federated rounds. Table 6.1 summarizes aggregated (aggregate/mean) Adjusted R2 scores and test loss returned by individual clients for every round in FL.

The FedAvg algorithm selects the clients for training and testing using the client-picking strategy detailed in Section 4.3.3. The Pytorch-LSTM model (Section 5.3) implemented for each client performs 2000 local epochs. We only conduct three federated rounds to achieve the necessary convergence rate. Additionally, we observe that the FedAvg algorithm works effectively with non-IID data.

In general, the accuracy of a regression model will be greater if its MSE, RMSE, and MAE values are lower, as discussed in Section 4.5. Our working system yields this characteristic. The results summarized in Table 6.1 indicate that as the number of FL rounds increases, FL performance improves and becomes more stable.

Table 6.1: FL system performance for different training and evaluation splits

# Training Clients	# Evaluation Clients	Aggregated Adjusted R2-score FL rounds			Aggregated Test loss FL rounds		
		1	2	3	1	2	3
1000	1000	-8.6688	0.8521	0.8457	0.010427	0.000269	0.000230
1000	500	-0.2704	0.7343	0.8597	0.001779	0.000289	0.000228
1000	100	0.1157	0.8636	0.9004	0.001610	0.000286	0.000223
500	1000	-1.0614	0.5684	0.8259	0.002440	0.000378	0.000245
500	500	-3.8580	0.7485	0.7187	0.005773	0.000321	0.000237
500	100	3.8839	0.7769	0.6865	0.004482	0.000353	0.000259
100	1000	-2.6703	0.8524	0.6808	0.004724	0.000240	0.000336
100	500	-0.5017	0.7263	0.8298	0.001554	0.000489	0.000359
100	100	-1.9932	0.6750	0.8293	0.006279	0.000398	0.000271

We take the two best-obtained results from the above experiments (Table 6.1) and discuss the training, test loss, and other regression metrics in detail.

1. 1000 training; 500 evaluation clients
2. 1000 training; 100 evaluation clients

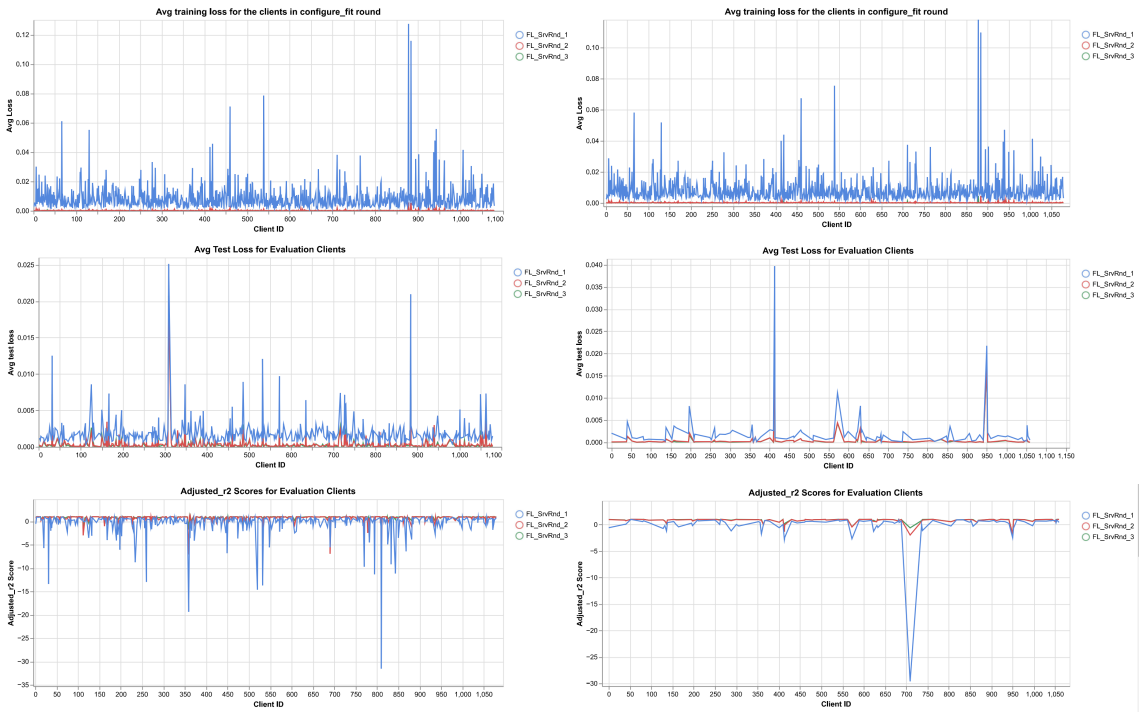

Figure 6.1: Train loss, Test loss, Adjusted R2 scores for 1000 training clients and 500 evaluation clients (left) and 100 evaluation clients (right)

Figure 6.1 illustrates how the training and test loss eventually decreases as the federated round progresses. Consequently, the Adjusted R2 scores increase, indicating a good convergence or prediction of the actual battery capacity for EVs. The other regression metrics, MSE, RMSE, and MAE, as illustrated in Figure 6.2 and Figure 6.3 for (1) and (2) the best-obtained results, respectively. We calculate them for each client during every round of FL. They gradually decrease as the number of federated iterations rises, demonstrating high performance with three FL iterations.

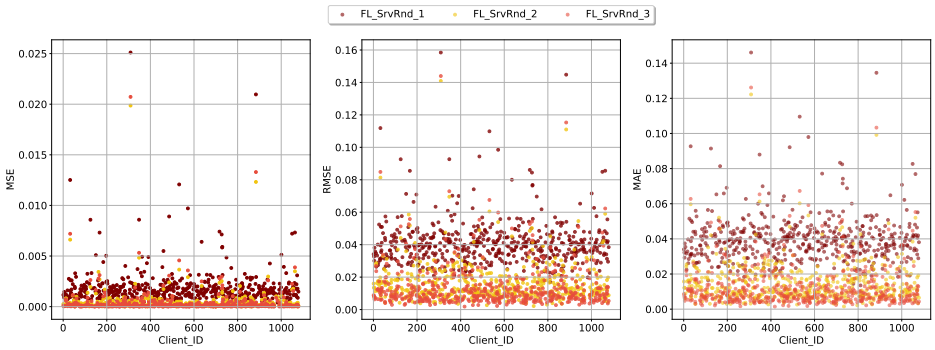

Figure 6.2: MSE, RMSE, and MAE scores for 500 evaluation clients after 3 federated rounds of training 1000 EV clients.

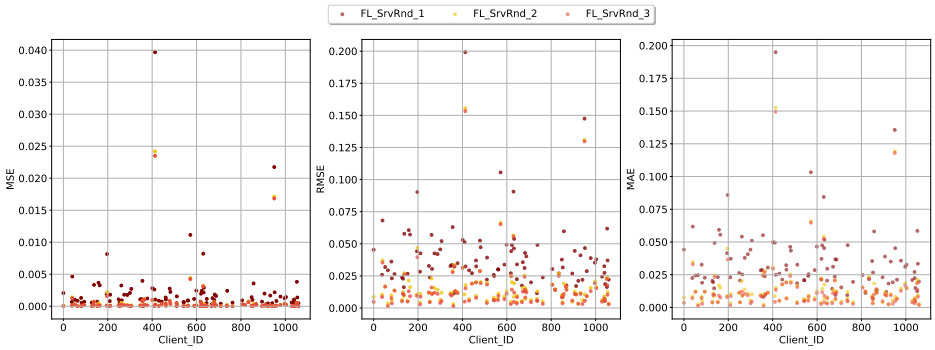

Figure 6.3: MSE, RMSE, and MAE scores for 100 evaluation clients after 3 federated rounds of training 1000 EV clients.

Each FL communication round represents one epoch/step of the global model, indicating that FL is an iterative process requiring communication rounds to achieve convergence. As shown in Figure 6.1, there are a few clients whose poor performance in the first round of FL training results in a precipitous decline in their Adjusted R2 scores. Eventually, however, the divergence in outcomes is reduced. Results from a subset of these clients showing this pattern are shown in Table 6.2.

Table 6.2: Selected clients' performance (From (1))

	Adjusted R2 scores		
	FL rounds		
	1	2	3
Client 810	-31.55	-9.98	-2.74
Client 474	-3.67	0.5053	0.8449
Client 190	-4.09	0.5340	0.9352
Client 359	-19.36	-7.07	-1.76
Client 234	-8.76	0.7177	0.7573

6.1.2 Performance of Federated Learning over Local Learning (Personalized Training)

Since FL involves multiple clients, it is a collaborative learning approach that utilizes larger shared data samples. As a result of differences in the underlying distributions of our clients in FL, we always observe a skewness in data distributions leading to more diverse labeled datasets. The local model, which trains using each client's local data, is referred to as the local baseline or personalized model.

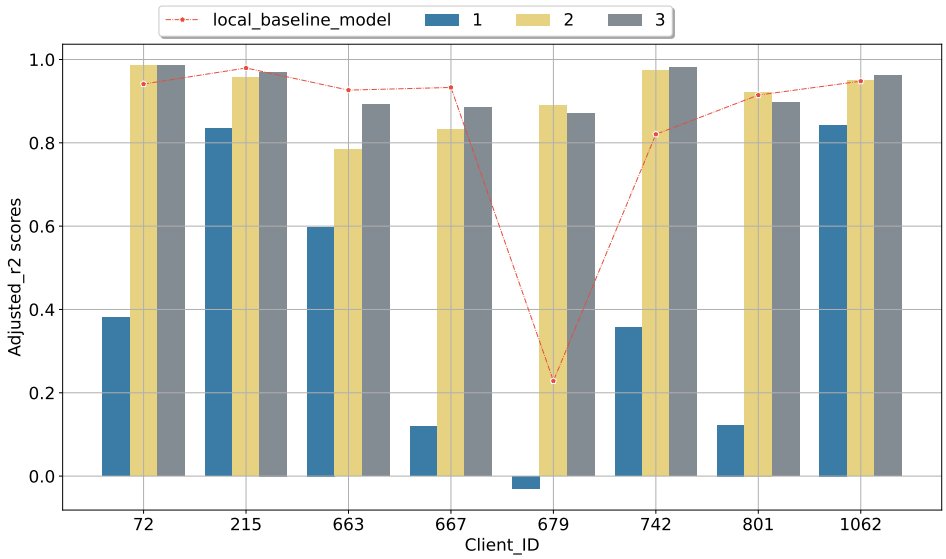

Figure 6.4: Adjusted R2 scores for different evaluation clients are compared to the local baseline model for 3 FL rounds with 100 EV clients as our training set.

Many clients contribute to the global FL model aggregation. This indicates that it receives more labeled data updates than the local baseline model. Training, validating, and testing a local model on its data may deliver beneficial results. However, this could have negative consequences for unknown or unseen data.

The data pool of an FL process is unseemingly high here since it involves global weight updates from thousands of EV clients with different speed and acceleration patterns during their mobility. In contrast, the personalized model is limited to a single EV's data. Thus it is possible to model a system with better robustness in FL, which can significantly assist generalization performance preserving user privacy.

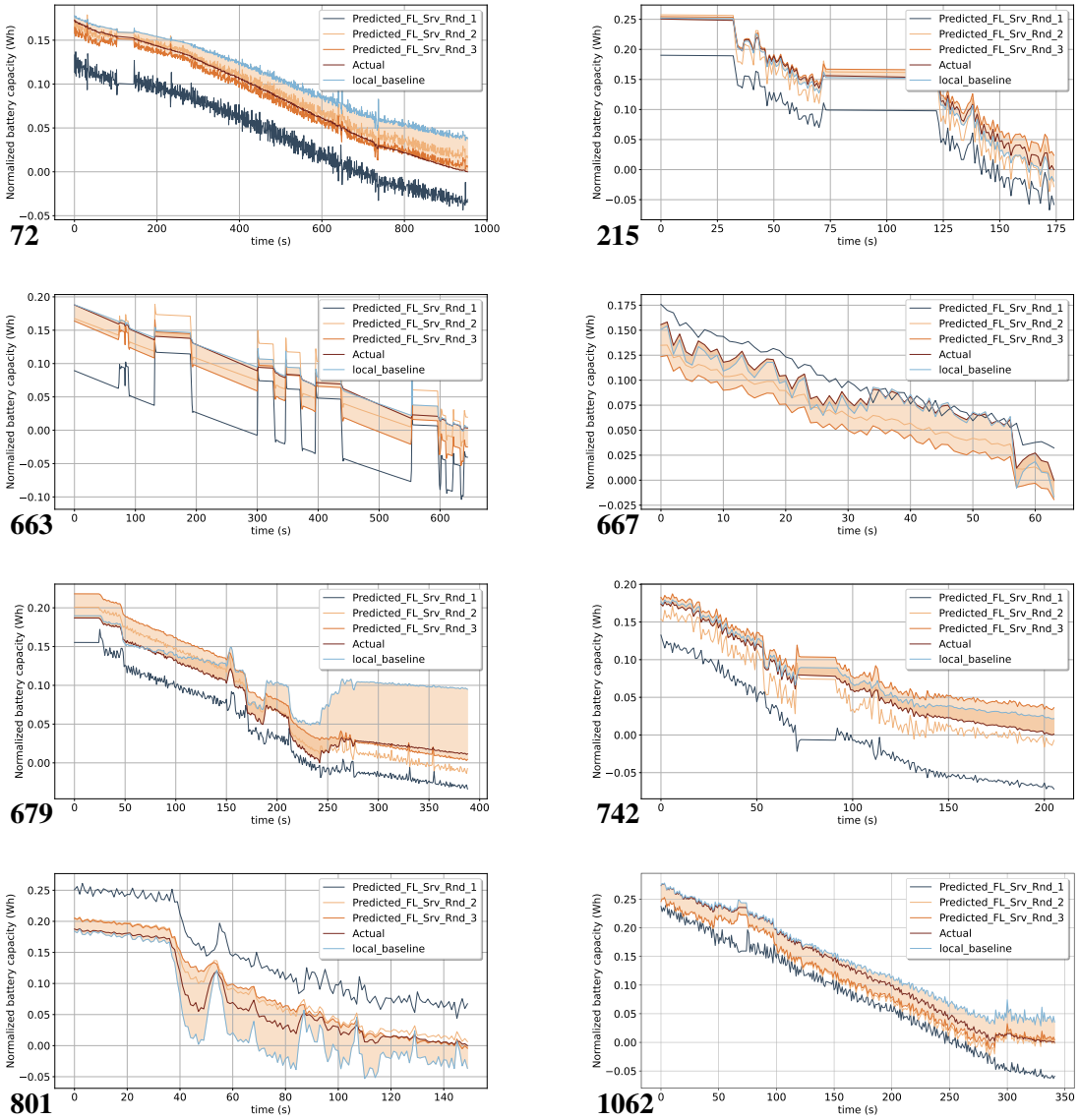

Figure 6.5: Comparison of the battery capacity estimation with the local baseline model, 3 FL rounds, and the actual ground truth for EV client IDs.

Our experiments investigate the accuracy by comparing local and FedAvg models to estimate the remaining battery capacity. Each EV trains its data without external knowledge in this baseline training scenario, which could lead to model redundancy. Though this type of training preserves privacy, it suffers from achieving robustness. The model's prediction goes down when a locally trained EV encounters a circumstance it has never seen before. On the other hand, the implemented

FedAvg is more promising than the local model since it receives global updates from 1000s of clients. The performance of different clients chosen randomly by the user and evaluated using the FedAvg from the Flower framework is shown in Figure 6.4. We observe that the prediction accuracy for most clients (72, 679, 742, 801, and 1062) in the third FL iteration is better than those of the local baseline model hence achieving a reasonable convergence rate.

Figure 6.5 illustrates the convergence rate of the implemented FL for eight randomly chosen Client IDs 72, 215, 663, 667, 679, 742, 801, and 1062. The remaining battery capacity in Wh (normalized values) is predicted for these eight clients. Here, we highlight the area between the third round of FL and the actual battery capacity value and between the local baseline model and the actual battery capacity value. We see a very high model divergence in the local baseline model performance for Client ID 679.

6.1.3 Training Time

As explained in Section 4.3.5, Flower wraps the VCE using Ray to configure and allocate the resources in the system. The runtime behavior during the training and evaluation phases of the FL system depends on CPU availability, memory space and speed, and the intensity of computation.

We use Ray's dashboard to determine the memory utilization during the FL process, which displays CPU utilization in addition to other valuable metrics. A sample view of the Ray dashboard is illustrated in Figure 6.6. The Flower simulation toolkit's API includes an option to activate the Ray dashboard and adjust the number of CPUs (num_cpus) used in the system, as demonstrated in the code snippet.

```
ray_init_args = {"include_dashboard": True}

fl.simulation.start_simulation(
    ................,
    client_resources={"num_cpus": 0.1},
    config=fl.server.ServerConfig(num_rounds=args.iterations),
    strategy=strategy,
    ray_init_args=ray_init_args,
 )
```

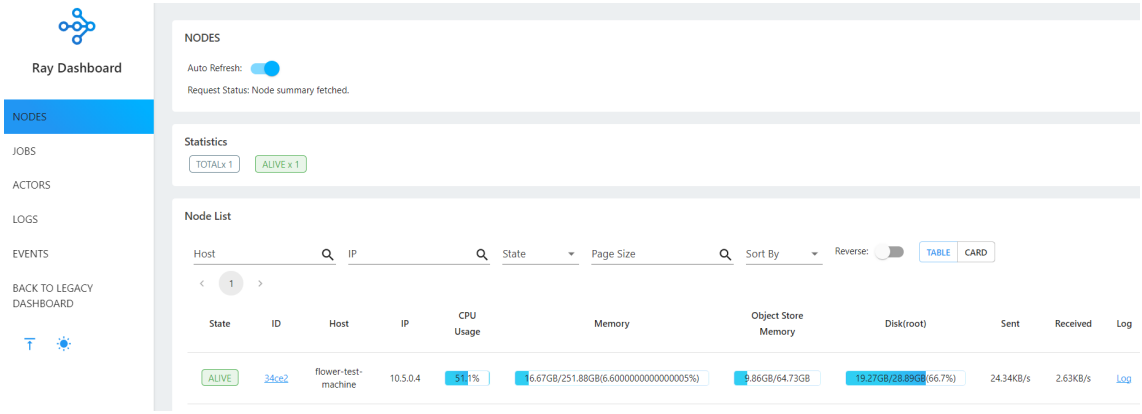

Figure 6.6: Ray Dashboard

As an alternative interactive process viewer, we use htop. Throughout the FL procedure, we use htop to monitor the load on each CPU core. This is useful for modifying the num_cpus in the Flower simulation API configuration option if the CPUs are underutilized or overutilized during the training process. Figure 6.7 depicts a representative illustration of CPU usage using htop.

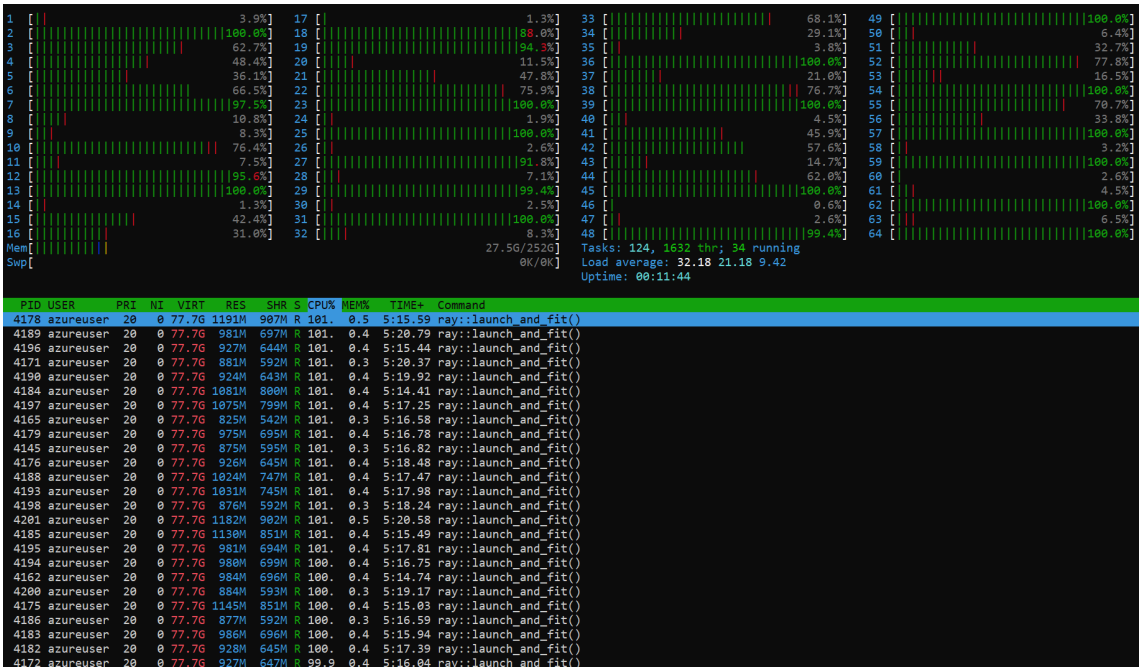

Figure 6.7: View of htop interactive system-monitor

Figure 6.8: No. of EV clients per round and the computation/runtime (in seconds) for 3 FL rounds.

The number of rounds is a key hyper-parameter in FL. It facilitates the determination of the required convergence rate for clients. Foretelling the time complexity of a system requires computing the time needed for each training and evaluation cycle. As an inference metric, this allows us to comprehend the FL training time using an energy-aware manner. In large simulation systems, runtime computation is helpful when specific modifications are necessary.

We determine the computation time for various EV clients throughout the training and evaluation phase. The implemented federated environment selects 1000 clients for training, sends model updates to the server, and evaluates 1000 clients in 3131.83 seconds, regarded as the upper bound of the training time for the implemented system. As the number of clients ascends, we see a linear increase in computation time. Figure 6.8 shows the computation time for different training and evaluation client ratio computed over three FL iterations.

6.1.4 Experiment Tracking using W&B

As described in Section 4.4, the integration of W&B with the implemented FL system enables us to monitor large-scale EV experiments. We utilize the various W&B components to migrate the FL system to a production-ready environment. As the FL training commences, we determine the model performance metrics for thousands of clients and use W&B's interactive dashboard features to comprehend their performance. The model artifacts are also stored in the project workspace configurations. Following is an illustration study of the integration of W&B with our implemented system. The main blocks of the W&B integrated with the current FL system are as shown below,

FL iteration groups: As FL is an iterative method, tracking each round's convergence results is essential to determine when training should conclude. We can use W&B's 'group' feature to group the runs in such situations. Figure 6.9 depicts the orderly grouping of the various FL rounds. Each round of FL will have its metrics. Thus, differentiating the various obtained results from the grouped FL rounds facilitates the end user's ability to configure the FL environment accordingly.

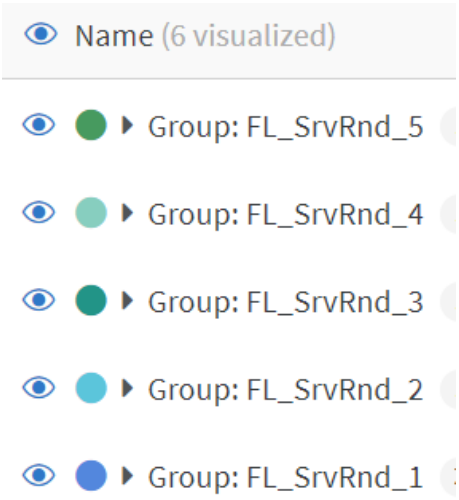

Figure 6.9: Configuring rounds in W&B

Custom Charts: Recording custom charts in W&B improves the user's visualization. Additionally, W&B's custom charts are highly interactive. Incorporating and sharing any graph the user requires with project collaborators is simpler. There is a slider option for all generated graphs. In Figure 6.10 (a), for instance, a slider displays the client ID 696's Adjusted R2 scores for various rounds in FL. Additionally, it is possible to export custom charts to various formats. Figure 6.10 (b) depicts a matplotlib-generated graph easily embeddable in W&B.

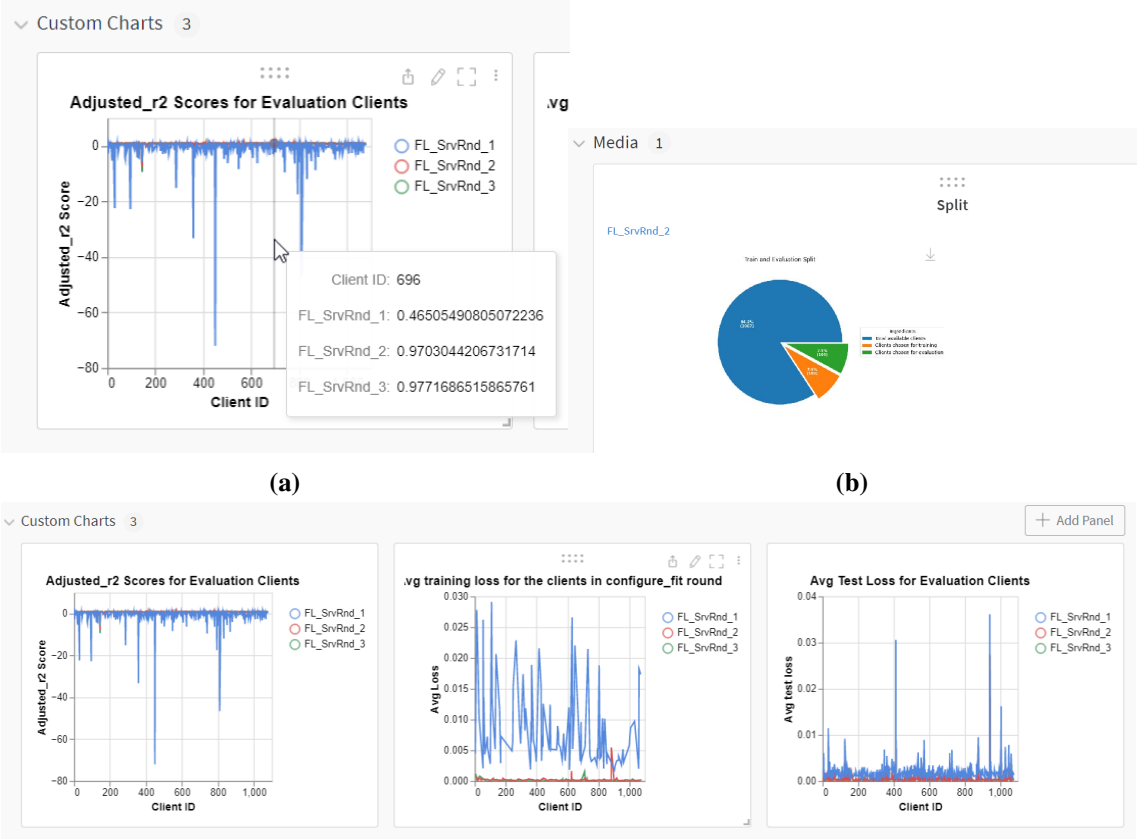

Figure 6.10: Logging custom charts in W&B

Tables: Tables are another feature of W&B. They assist in sorting, filtering, and grouping the various metrics for clients in FL. They also facilitate logging tables from multiple runs and the subsequent comparison of results in the project workspace. They are adaptable because we can export only the required client ID information to a CSV file. Figure 6.11 illustrates a sample representation of the generated table in W&B.

System details: In order to design a resource-aware system during the FL simulation process, it is essential to keep track of various system resource data. Keeping track of the system's specifics is a crucial feature of W&B. W&B automatically logs system metrics such as Network traffic (sent and received bytes), system memory usage, disk I/O usage, and CPU threads usage every 2 seconds, averaging over 30 seconds. Since these metrics are recorded for each round of FL, they facilitate the development of a more adaptable resource-aware system. Figure 6.12 illustrates the logging of the system details by W&B.

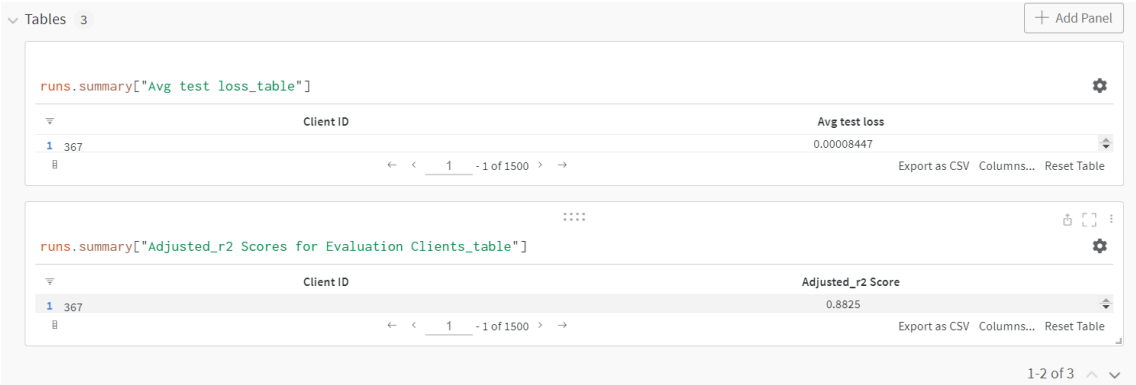

Figure 6.11: Tables generated in W&B

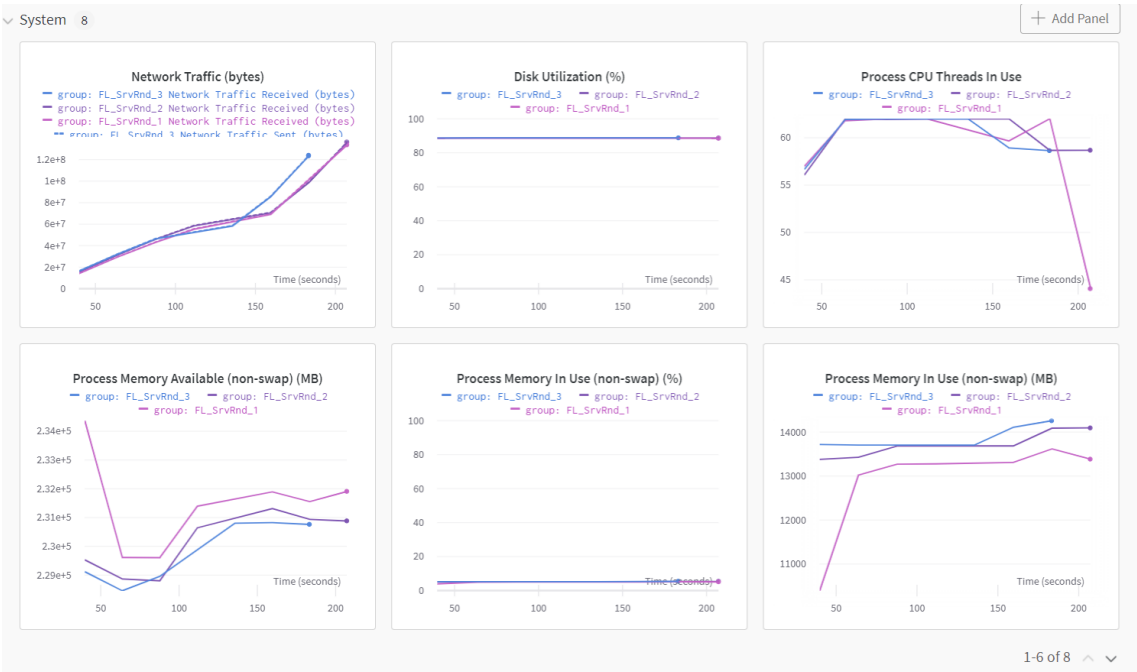

Figure 6.12: System details generated in W&B

Artifacts: The W&B artifacts are used to version the various FL rounds and experiments. The artifact's checksum allows us to comprehend the changes and track the new version of the training. Figure 6.13 illustrates an example of artifact generation for various FL rounds.

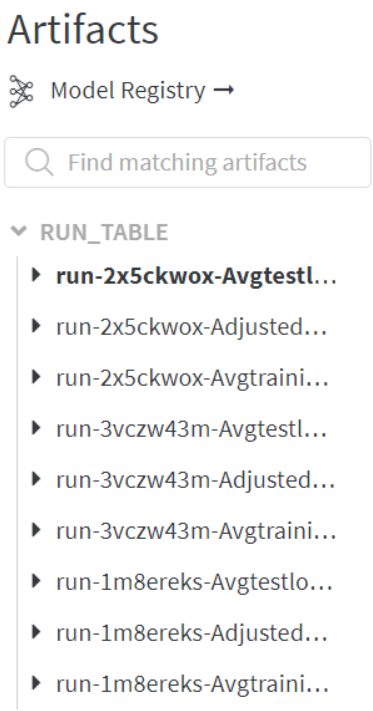

Figure 6.13: Artifacts in W&B

6.1.5 Differential Privacy

As described in the Section 2.1.5 and Section 4.3.6, we add noise to each local EV model to prevent model updates (weights/gradients) against attack or data leakage. The added noise aids in determining the relationship between accuracy and privacy budget epsilon (ϵ).

We select ten clients for the training using the Client Picking Strategy: 50, 76, 136, 186, 539, 676, 965, 979, 1019, and 1039. The PrivacyEngine class of Opacus is attached to these clients. Each local client model is trained for 2000 local epochs with a learning rate of 0.01 for all clients. We set the minimum acceptable value for δ to 1e-3. The various noise levels selected fall within the set of $\sigma = \{0.5, 1.0, 1.5, 2.0\}$. The clients selected for determining the accuracy of an end-to-end DP model's implementation are 77, 133, 545, and 1049.

As depicted in Figure 6.14, the value of epsilon increases as the noise value decreases. As the epsilon value decreases, the model's privacy protection improves. Here, for a noise value of 1.5 and 2.0, we observe that the epsilon value is near zero for almost all clients. On the contrary, introducing noise affects the model performance. As illustrated in Figure 6.15, we see a drastic decrease in the Adjusted R2 scores with an increase in noise.

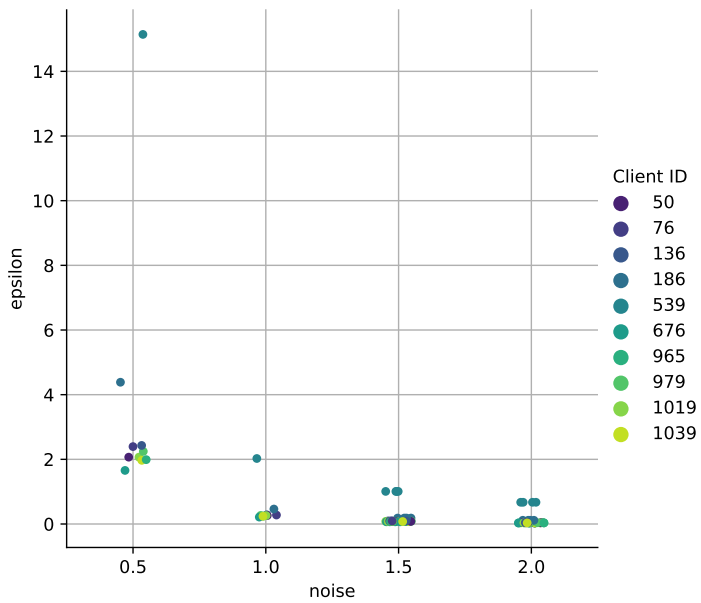

Figure 6.14: Epsilon values for different noise multipliers

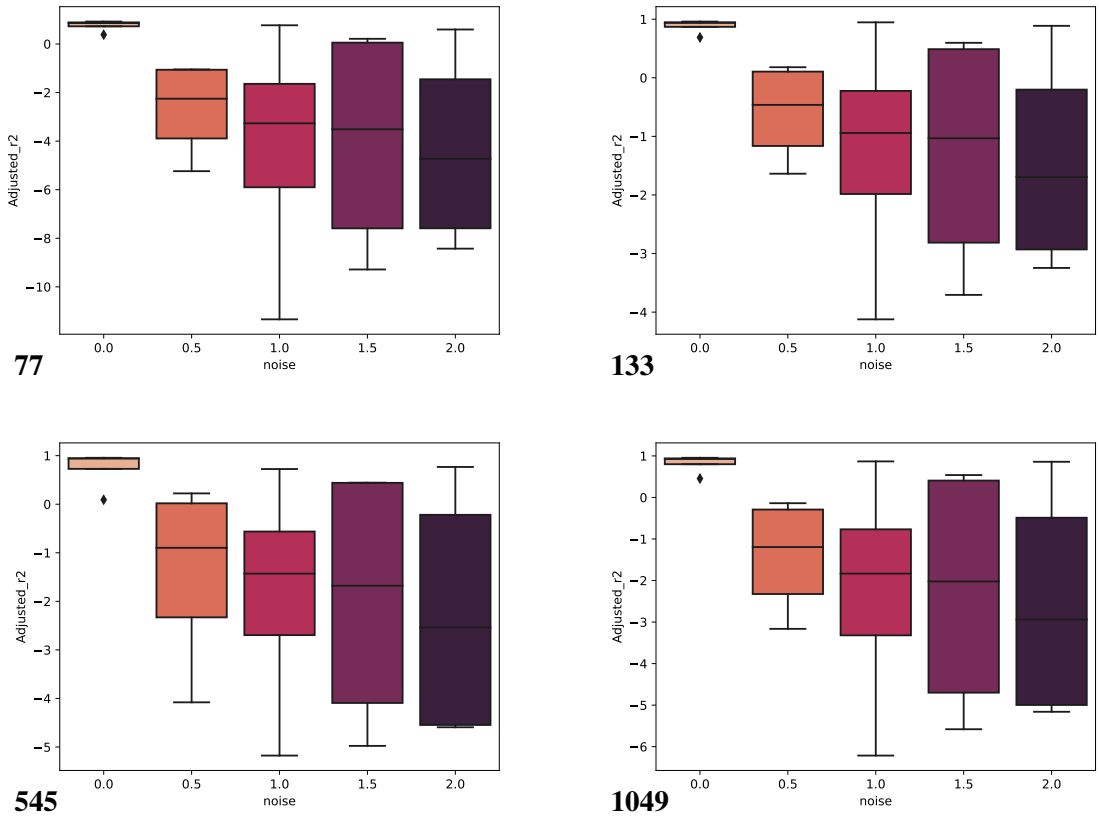

Figure 6.15: Adjusted R2 scores for different noise multipliers

6.2 Performance Results Use-case (b): Energy-demand forecasting for CSs

In our Use-case (b), we calculate the energy transferred by each CS to various vehicles. We implement the FaultTolerantFedAvg algorithm to address faulty clients, as described in Section 4.3.1. Once we determine that a CS does not have enough data to begin training, we label it a faulty client. Therefore, we eliminate these clients during training and only collect updates from clients with sufficient samples. This method ensuring a fault-tolerance mechanism is an enhanced version of the FedAvg.

The three outcomes mentioned below suggest that we have a reliable forecasting model. In our experiments, the energy consumption forecast accuracy for all CSs is above 90 percent, indicating a good model prediction, as shown in Figure 6.16. Figures 6.17 and 6.18 also show a gradual decline in MSE and RMSE scores as the FL round progresses, indicating superior model performance.

Figure 6.19 illustrates the convergence rate of the implemented FL for CS CID_Sind_cs_18.

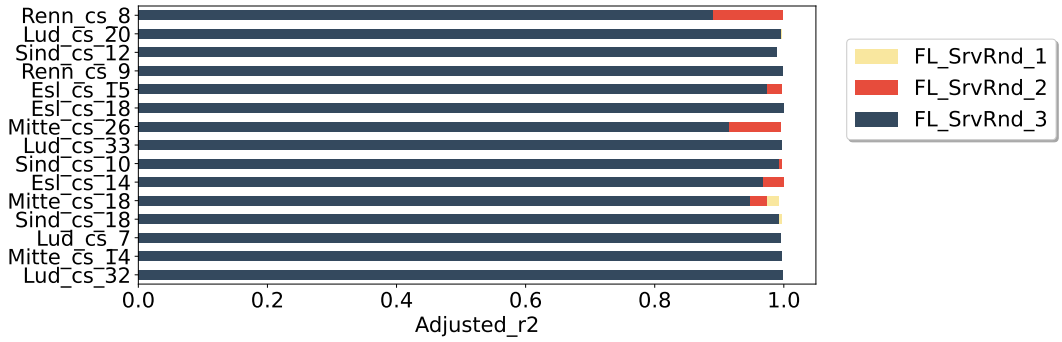

Figure 6.16: Adjusted R2 for 15 CSs after 3 FL rounds

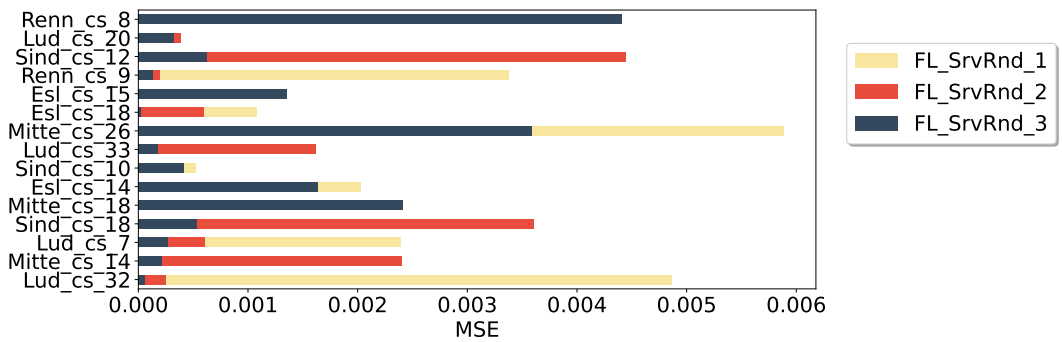

Figure 6.17: MSE for 15 CSs after 3 FL rounds

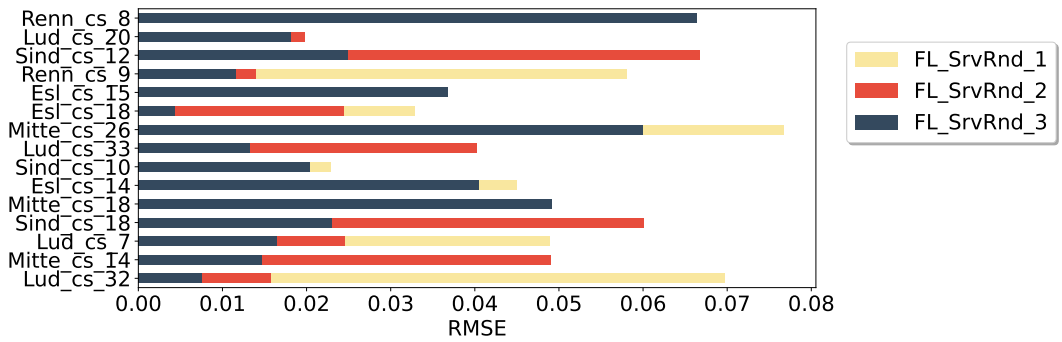

Figure 6.18: RMSE for 15 CSs after 3 FL rounds

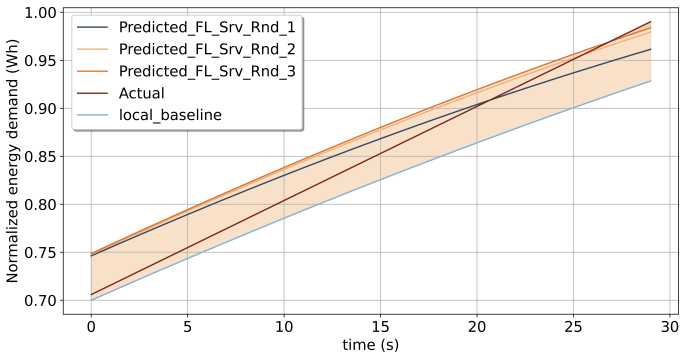

Figure 6.19: Comparison of the energy-demand with the local baseline model, 3 FL rounds, and the actual ground truth of the CS CID_Sind_cs_18

6.3 Discussion

In this section, we examine the obtained results in light of the research questions and justify the outcomes of the developed FL system.

RQ1 : How can we incorporate the FL environment justifying user privacy and system heterogeneity to realize use-cases (a) and (b)?

The cumulative outcomes of Sections 6.1 and 6.2 serve as the foundation for this research question. The Flower framework was used to implement a federated environment for the two use cases. Flower's single-node simulation enables us to configure the FL environment without disclosing client information, hence protecting user privacy as intended.

Use-case (a):
As demonstrated in Figure 6.1, as the FL iteration advances, the training and test loss for 1000 training clients and 500/100 evaluation clients decreases. Table 6.1 also demonstrates this characteristic. For instance, the aggregated Adjusted R2 score for 1000 training and 500 evaluation clients increases from -0.2704 at the end of the first round of FL to 0.8597 at the end of the third FL round. Figures 6.2 and 6.3 illustrate the decrease in MSE and RMSE from the first to the third round of FL. As shown in Table 6.2, despite the fact that specific clients demonstrate relatively poor performance in the initial FL round, the Adjusted R2 scores rapidly improve in successive FL rounds. In this use case we assume that no client is faulty.

As seen in Figure 6.4, the implemented FL model's performance is higher compared to the local model (personalized model) for most of the clients. When a client trains its data using its local model, its ability to handle unseen data becomes less plausible. As depicted in Figure 6.5, multiple clients in FL provide converging results for the battery capacity prediction compared to the local baseline model. This means that the developed FL system can outperform the local baseline model in several scenarios due to the model updates from the non-IID data across thousand of clients.

Use-case (b):
As shown in Figure 6.16, the Adjusted R2 score for the vast majority of CSs is considerably over 0.9, indicating a model with a high level of accuracy. Similarly, Figures 6.17 and 6.18 demonstrate that as the FL iterations advance, the MSE and RMSE results for 15 CSs for three FL rounds fall steadily. Again, this demonstrates the FL model's stability.

The previous result discussion provides a viable and practicable method for protecting the privacy of individual users of EVs and CSs in a diverse automotive environment (simulated non-IID data, Section 5.2). Utilizing the FedAvg technique to interpret the convergence findings allows us to conclude that the implemented system handles production-grade FL privacy and statistical data heterogeneity adequately. Thus, all the above results depict that the implemented system has a high convergence rate for the production-ready automotive environment.

RQ2 : How does the above proposed system system provide incentives for deploying FL in a production-ready automotive environment?

RQ2.1 How does the system handle the training and evaluation of numerous clients?
During each iteration of the FL process, as depicted in Table 6.1, we train and evaluate 1000 EV clients in the Stuttgart region based on simulation results. In conjunction with Ray, Flower's VCE enables us to schedule a large-scale simulation in a resource-aware manner. As described in Section 4.3.5, the ClientManager and ClientProxy classes of Flower facilitate the generation of client instances only when they are selected for training and evaluation. This resource-aware instantiation provided by Flower, mitigates the overhead of resources and enable large-scale simulation for a production environment.

As seen in Figure 6.6, the dashboard visualization is an interactive representation that enables the user to tailor simulation resources as required. As indicated in Figure 6.7, Ray enables us to alter CPU core utilization based on availability, reducing CPU consumption overhead. Figure 6.8's upper bound of computing time for the FL system reveals that 1000 EV training and evaluation clients, each with a Pytorch-LSTM model (Section 5.3) and 2000 local epochs, require approximately 3131.83 seconds to complete training and evaluation. Consequently, establishing a time constraint for the debugging and deployment of large-scale automotive simulations.

RQ2.2 How do we implement a user-friendly client-picking strategy for active client contribution?
As described in Section 4.3.3, the FedAvg algorithm is modified to pick the required clients for the training and evaluation phase. For instance, as shown in Figure 6.4, we pass the eight client IDs who are interested to know their actual battery capacity prediction outcomes.

Such a selection method is required to comprehend the clients who are currently utilizing the system and those who have departed. This enables active contribution instead of arbitrarily and coercively picking clients for training or evaluation. Thus, it provides a client-friendly production-grade interface by allowing the client to choose whether to participate in the process and eventually mitigates the communications costs.

RQ2.3 How can we handle faulty clients during the training and evaluation process?
We have implemented FaultTolerantFedAvg, an extension of the FedAvg algorithm designed to accommodate malfunctioning clients as in Section 4.3.1. By employing this algorithm, we ensure that the FL system handles errors more effectively. It also specifies that when an FL training round encounters faulty or error-prone clients, it excludes them and collects updates from the remaining clients rather than halting the process altogether. By adjusting min_completion_rate_fit and min_completion_rate_evaluate in the algorithm, it is possible to achieve the desired outcomes, as seen in Figure 6.16.

In the production-grade automotive industry, FaultTolerantFedAvg outperforms FedAvg by recognizing defective clients in the system. Due to the dynamic nature of EVs and CSs, a decline in participation is commonplace in vehicular networks. Consequently, utilizing FaultTolerantFedAvg as an extension to FedAvg improves the model aggregation outcomes in such scenarios.

RQ2.4 How can we integrate the proposed FL system with a production-grade MLOps platform for experiment tracking?
A production-grade automotive FL system must provide model tracking, deployment, and interactive possibilities using an MLOps platform for easy production. We do this by integrating the FL model to W&B.

As seen in Figures 6.9 and 6.10, we can track the development of results across the FL rounds. This experiment tracking aids us in examining large-scale simulations. The specific clients who experience inconsistent results or discrepancies in model performance are discovered and can be instantly notified. The complete network traffic and system resource usage can be observed and understood graphically in W&B as in Figure 6.12. Figures 6.11 and 6.13 also provide numerous other data storage possibilities in W&B, such as table generation and artifact tracking.

The APIs provided by W&B are easily integrated with the Flower framework to track the progress and for interactive visualizations. Also, the collaboration of results between the client and the central server results in amicable production-grade environment development. Thus, W&B offers model versioning, result visualization, and report generation for the FL model in a production-grade environment.

RQ3 : How can we additionally enhance the privacy guarantees for the local client model during weight updates using differential privacy?

In a production-level system, when chosen for training, each client is assigned the PrivacyEngine class from the Opacus library. We can construct a (ϵ, δ)-DP model by carefully adjusting the noise multiplier σ and clipping threshold (C) hyper-parameters sent to the PrivacyEngine class.

Figure 6.14 shows that epsilon values are highest for a noise multiplier of 0.5. As the noise multiplier increases, the epsilon value eventually declines. Consequently, a drop in epsilon value suggests a private model immune to external attacks. On the contrary, as seen in Figure 6.15, there is a decrease in the Adjusted R2 scores for four client IDs with noise injection into the local client models. This is a common trait of a DP model [98].

Providing an end-to-end DP model in a production-grade environment is crucial since EVs participating in the FL process are again susceptible to attacks while connecting with the central server, via V2X communication, for example. In the preceding experiments, we dealt with a superficial level of privacy for the local client model by establishing a uniform norm clipping and noise multiplier value in the interval [0.5, 2.0]. This permits using Opacus in Flower to develop a DP model for the automotive industry.

It should be emphasized, however, that in certain situations, each client may have distinct privacy requirements. For instance, a client may need less privacy and more precise results, or vice versa. Providing such continual adaptations of the privacy budget for each client results in server-side overhead during the global model's aggregation.

7 Conclusion and Outlook

7.1 Conclusion

In this work, we have implemented a Flower-based FL environment without disclosing individual client information. We have generated a heterogeneous dataset using the traffic and EV model in SUMO. The developed system can predict the battery life of EVs and the energy consumption of CSs in a specific region, addressing two widespread use cases in the automotive industry and thus answering the research question RQ1.

As in RQ2, the thesis emphasizes the creation of a system appropriate for use in a production environment. One such framework that is easily adaptable and scalable in a production environment is the Flower federated framework. Utilizing the design concept of the Flower, the implemented system deploys a large-scale simulation for 1000 EV clients in a resource-efficient manner (RQ2.1). The client selection technique permits active client participation without stagnant clients, demonstrating that Flower's software stack is user-friendly and allows customizations (RQ2.2). The implementation of system's fault tolerance helps discover defective clients without halting the entire FL process (RQ2.3). Flower's ability to integrate with W&B for experiment tracking makes it easy to configure in an MLOps platform for production, thus simplifying training outcomes, model artifact storage, and report generation (RQ2.4).

The implemented end-to-end DP model and its various hyper-parameters provide end-to-end privacy guarantees to EVs, thus answering research question RQ3.

The established system handles client-requested privacy and statistical data heterogeneity, thrusting the system to work in a production-grade environment. The discussed associated strategies and algorithms using a user-friendly Flower framework produce high-convergence findings, enabling the initial simulation phase toward FL production readiness in automotive use cases.

7.2 Limitations

The described simulation environment covers numerous challenges in FL and is a foundation for various system-level and algorithm research approaches. They demonstrate a given technique's applicability by outlining various real-world outcomes. Despite its benefits, it is vital to consider the limitations of such a system. The limitations are discussed as follows,

- In SUMO, we utilize the straightforward and generalized EV energy model. This model is constructed with a constant parameter for propulsion efficiency and regenerative braking [90], which may compromise the actual behavior of EV mobility characteristics such as acceleration and associated battery consumption during its motion. Thus, this impacts the overall battery life calculation.

- The currently implemented FL system follows a single-node simulation. Two core challenges in FL, the system heterogeneity and communication costs and overhead are addressed on a very minute scale because of the adoption of single-node simulation. In real-world FL scenarios, the communication rates could differ from one client to another, a missing aspect not addressed in single-node simulation. Additionally, each client may have unique system resources, which is a concern that must be managed.

7.3 Future Work

Following the discussions and limitations, we suggest possible future improvements and extensions for the FL production-ready automotive system,

- Though precise data acquisition is not the focus of the thesis, it would be beneficial to extend the present SUMO model to account for other factors affecting the battery life of EVs. There is ongoing research in this direction to simulate more realistic data using the SUMO simulator [49, 53].
- Flower supports the seamless transition from single-node simulation to multi-node simulation and deployment on edge devices. We can leverage this attribute of Flower and address problems such as system heterogeneity and communication costs.

  We can further deploy the implemented model in the EVs in the real-time scenario. The instantaneous data can be collected and collaborated from the Vehicle Control Unit (VCU) and the Battery Management System (BMS) in the car. The data can then be routed to a computing device place inside the vehicle, such as a Raspberry Pi or NVIDIA Jetson Nano, with sufficient CPU and GPU capabilities for training the model. The trained weights can then be updated on the central server via Flower's RPC communication mechanism. This is a future potential edge deployment of the current system.
- For our use cases, we have implemented the FedAvg and FaultTolerantFedAvg algorithms. Despite the high convergence rate for the majority of clients, there are a few clients whose performance must be enhanced due to their very heterogeneous data. In such circumstances, the FedAvg algorithm can be extended to the FedProx algorithm for more robust convergence.
- The end-to-end DP model developed is only evaluated for the gradient-norm clipping value C=1.0. The value of C affects the gradient's orientation. If not chosen accurately, it could lead to divergence in results. Leveraging the hyperparameter sweep functionality in W&B, we can tune a better value for the gradient norm clipping for a given noise multiplier. The adaptive clipping value selection can be one of the future attempts to address the trade-off between obtained accuracy and noise introduced into the system.

Bibliography

[1] Martin Abadi et al. "Deep Learning with Differential Privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, October 2016. DOI: 10.1145/2976749.2978318. URL: https://doi.org/10.1145%2F2976749.2978318 (page 24).

[2] Daniel J Beutel et al. "Flower: A Friendly Federated Learning Research Framework". In: *arXiv preprint arXiv:2007.14390* (2020) (pages 29, 30, 32, 36).

[3] Daniel J. Beutel et al. *Flower: A Friendly Federated Learning Research Framework*. arXiv:2007.14390 [cs, stat] version: 1. July 2020. URL: http://arxiv.org/abs/2007.14390 (visited on July 6, 2022) (page 30).

[4] Daniel J. Beutel et al. *Flower: A Friendly Federated Learning Research Framework*. en. arXiv:2007.14390 [cs, stat]. March 2022. URL: http://arxiv.org/abs/2007.14390 (visited on July 6, 2022) (pages 30, 31).

[5] Subrato Bharati et al. "Federated learning: Applications, challenges and future directions". In: *International Journal of Hybrid Intelligent Systems* 18.1-2 (May 2022), pages 19–35. DOI: 10.3233/his-220006. URL: https://doi.org/10.3233%2Fhis-220006 (pages 19, 20, 23).

[6] Lukas Biewald. *Experiment Tracking with Weights and Biases*. Software available from wandb.com. 2020. URL: https://www.wandb.com/ (pages 26, 53).

[7] Benedikt Buhk, Rasmus Rettig. "Simulation based method for the analysis of energy-efficient driving algorithms using SUMO". In: November 2021 (page 39).

[8] Sebastian Caldas et al. *LEAF: A Benchmark for Federated Settings*. 2018. DOI: 10.48550/ARXIV.1812.01097. URL: https://arxiv.org/abs/1812.01097 (page 29).

[9] Jianlong Chen et al. "An Improved Gated Recurrent Unit Neural Network for State-of-Charge Estimation of Lithium-Ion Battery". In: *Applied Sciences* 12 (February 2022), page 2305. DOI: 10.3390/app12052305 (page 25).

[10] Junbin Chen et al. "Federated Learning for Bearing Fault Diagnosis with Dynamic Weighted Averaging". In: *2021 International Conference on Sensing, Measurement & Data Analytics in the era of Artificial Intelligence (ICSMD)*. 2021, pages 1–6. DOI: 10.1109/ICSMD53520.2021.9670854 (page 28).

[11] Alexander Chowdhury et al. "A Review of Medical Federated Learning: Applications in Oncology and Cancer Research". In: July 2022, pages 3–24. ISBN: 978-3-031-08998-5. DOI: 10.1007/978-3-031-08999-2_1 (page 21).

[12] Marcos F. Criado et al. *Non-IID data and Continual Learning processes in Federated Learning: A long road ahead*. 2021. DOI: 10.48550/ARXIV.2111.13394. URL: https://arxiv.org/abs/2111.13394 (pages 23, 62).

[13] Dimitrios Dimitriadis et al. *FLUTE: A Scalable, Extensible Framework for High-Performance Federated Learning Simulations*. 2022. DOI: 10.48550/ARXIV.2203.13789. URL: https://arxiv.org/abs/2203.13789 (page 30).

[14] Zhaoyang Du et al. "Federated Learning for Vehicular Internet of Things: Recent Advances and Open Issues". In: *IEEE Open Journal of the Computer Society* 1 (2020), pages 45–61. DOI: 10.1109/OJCS.2020.2992630 (page 20).

[15] Cynthia Dwork, Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Found. Trends Theor. Comput. Sci.* 9.3–4 (August 2014), pages 211–407. ISSN: 1551-305X. DOI: 10.1561/0400000042. URL: https://doi.org/10.1561/0400000042 (pages 23, 24).

[16] Cynthia Dwork et al. "Calibrating Noise to Sensitivity in Private Data Analysis". In: *Theory of Cryptography*. Edited by Shai Halevi, Tal Rabin. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pages 265–284. ISBN: 978-3-540-32732-5 (page 24).

[17] Cynthia Dwork et al. "Our Data, Ourselves: Privacy Via Distributed Noise Generation". In: *Advances in Cryptology - EUROCRYPT 2006*. Edited by Serge Vaudenay. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pages 486–503. ISBN: 978-3-540-34547-3 (page 24).

[18] Ahmet M. Elbir et al. *Federated Learning in Vehicular Networks*. 2020. DOI: 10.48550/ARXIV.2006.01412. URL: https://arxiv.org/abs/2006.01412 (page 28).

[19] Robin C. Geyer, Tassilo Klein, Moin Nabi. *Differentially Private Federated Learning: A Client Level Perspective*. 2017. DOI: 10.48550/ARXIV.1712.07557. URL: https://arxiv.org/abs/1712.07557 (page 24).

[20] Flower Github. *Flower Github*. URL: https://github.com/adap/flower (page 31).

[21] SUMO Github. *SUMO Github*. URL: https://github.com/eclipse/sumo (pages 34, 42).

[22] Juan Jesús Gonzalez-Delicado et al. "Alicante-Murcia Freeway Scenario: A High-Accuracy and Large-Scale Traffic Simulation Scenario Generated Using a Novel Traffic Demand Calibration Method in SUMO". In: *IEEE Access* 9 (2021), pages 154423–154434. DOI: 10.1109/ACCESS.2021.3126269 (page 34).

[23] Yinghao Guo et al. "Efficient and flexible management for industrial Internet of Things: A federated learning approach". In: *Computer Networks* 192 (2021), page 108122. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.108122. URL: https://www.sciencedirect.com/science/article/pii/S1389128621001961 (page 19).

[24] Andrew Hard et al. *Federated Learning for Mobile Keyboard Prediction*. 2018. DOI: 10.48550/ARXIV.1811.03604. URL: https://arxiv.org/abs/1811.03604 (pages 15, 21).

[25] Chaoyang He et al. *FedML: A Research Library and Benchmark for Federated Machine Learning*. 2020. DOI: 10.48550/ARXIV.2007.13518. URL: https://arxiv.org/abs/2007.13518 (page 30).

[26] Philipp Heisig et al. "Bridging the Gap between SUMO & Kuksa: Using A Traffic Simulator for Testing Cloud-based Connected Vehicle Services". In: August 2019. DOI: 10.29007/9kkv (pages 34, 36).

[27] Sepp Hochreiter, Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (November 1997), pages 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. eprint: https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf. URL: https://doi.org/10.1162/neco.1997.9.8.1735 (page 25).

[28] Li Huang et al. "Patient clustering improves efficiency of federated machine learning to predict mortality and hospital stay time using distributed electronic medical records". In: *Journal of Biomedical Informatics* 99 (2019), page 103291. ISSN: 1532-0464. DOI: https://doi.org/10.1016/j.jbi.2019.103291. URL: https://www.sciencedirect.com/science/article/pii/S1532046419302102 (page 21).

[29] Xixi Huang et al. "DP-FL: a novel differentially private federated learning framework for the unbalanced data". In: *World Wide Web* 23 (July 2020). DOI: 10.1007/s11280-020-00780-4 (page 25).

[30] Jichu Jiang et al. "Federated Learning in Smart City Sensing: Challenges and Opportunities". In: *Sensors* 20 (October 2020), page 6230. DOI: 10.3390/s20216230 (page 15).

[31] Amelia Jiménez-Sánchez et al. *Memory-aware curriculum federated learning for breast cancer classification*. 2021. DOI: 10.48550/ARXIV.2107.02504. URL: https://arxiv.org/abs/2107.02504 (page 21).

[32] Peter Kairouz et al. *Advances and Open Problems in Federated Learning*. 2021. arXiv: 1912.04977 [cs.LG] (pages 15, 19, 20, 30).

[33] Sai Praneeth Karimireddy et al. *SCAFFOLD: Stochastic Controlled Averaging for Federated Learning*. 2019. DOI: 10.48550/ARXIV.1910.06378. URL: https://arxiv.org/abs/1910.06378 (page 22).

[34] Ivan Kholod et al. "Open-Source Federated Learning Frameworks for IoT: A Comparative Review and Analysis". In: *Sensors* 21.1 (2021). ISSN: 1424-8220. DOI: 10.3390/s21010167. URL: https://www.mdpi.com/1424-8220/21/1/167 (page 20).

[35] Lucas Koch et al. "Accurate Physics-Based Modeling of Electric Vehicle Energy Consumption in the SUMO Traffic Microsimulator". In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. Indianapolis, IN, USA: IEEE Press, 2021, pages 1650–1657. DOI: 10.1109/ITSC48978.2021.9564463. URL: https://doi.org/10.1109/ITSC48978.2021.9564463 (page 42).

[36] Jakub Konečný et al. "Federated Learning: Strategies for Improving Communication Efficiency". In: *NIPS Workshop on Private Multi-Party Machine Learning*. 2016. URL: https://arxiv.org/abs/1610.05492 (page 15).

[37] Tamás Kurczveil, Pablo Álvarez López, Eckehard Schnieder. "Implementation of an Energy Model and a Charging Infrastructure in SUMO". In: *SUMO*. 2014 (page 39).

[38] Fan Lai et al. "FedScale: Benchmarking Model and System Performance of Federated Learning at Scale". In: *International Conference on Machine Learning (ICML)*. 2022 (page 29).

[39] Fan Lai et al. "Oort: Efficient Federated Learning via Guided Participant Selection". In: *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 2021 (page 29).

[40] Kwing Hei Li et al. "Secure Aggregation for Federated Learning in Flower". In: *Proceedings of the 2nd ACM International Workshop on Distributed Machine Learning*. DistributedML '21. Virtual Event, Germany: Association for Computing Machinery, 2021, pages 8–14. ISBN: 9781450391344. DOI: 10.1145/3488659.3493776. URL: https://doi.org/10.1145/3488659.3493776 (page 31).

[41] Li Li et al. “A review of applications in federated learning”. In: *Computers & Industrial Engineering* 149 (2020), page 106854. ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2020.106854. URL: https://www.sciencedirect.com/science/article/pii/S0360835220305532 (pages 15, 21).

[42] Qinbin Li et al. “A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection”. In: *IEEE Transactions on Knowledge and Data Engineering* (2021), pages 1–1. DOI: 10.1109/TKDE.2021.3124599 (page 22).

[43] Tian Li et al. *Fair Resource Allocation in Federated Learning*. 2019. DOI: 10.48550/ARXIV.1905.10497. URL: https://arxiv.org/abs/1905.10497 (page 23).

[44] Tian Li et al. “Federated Learning: Challenges, Methods, and Future Directions”. In: *IEEE Signal Processing Magazine* 37.3 (May 2020), pages 50–60. DOI: 10.1109/msp.2020.2975749. URL: https://doi.org/10.1109%2Fmsp.2020.2975749 (pages 15, 22).

[45] Tian Li et al. *Federated Optimization in Heterogeneous Networks*. 2018. DOI: 10.48550/ARXIV.1812.06127. URL: https://arxiv.org/abs/1812.06127 (pages 22, 23).

[46] Xiang Li et al. *On the Convergence of FedAvg on Non-IID Data*. 2019. DOI: 10.48550/ARXIV.1907.02189. URL: https://arxiv.org/abs/1907.02189 (page 22).

[47] Xiaoxiao Li et al. *Multi-site fMRI Analysis Using Privacy-preserving Federated Learning and Domain Adaptation: ABIDE Results*. 2020. DOI: 10.48550/ARXIV.2001.05647. URL: https://arxiv.org/abs/2001.05647 (page 21).

[48] Wei Yang Bryan Lim et al. *Federated Learning in Mobile Edge Networks: A Comprehensive Survey*. 2019. DOI: 10.48550/ARXIV.1909.11875. URL: https://arxiv.org/abs/1909.11875 (page 15).

[49] Mingming Liu. “Fed-BEV: A Federated Learning Framework for Modelling Energy Consumption of Battery Electric Vehicles”. In: (August 2021) (pages 16, 27, 88).

[50] Sin Kit Lo et al. “A Systematic Literature Review on Federated Machine Learning”. In: *ACM Computing Surveys* 54.5 (June 2022), pages 1–39. DOI: 10.1145/3450288. URL: https://doi.org/10.1145%2F3450288 (page 15).

[51] Pablo Alvarez Lopez et al. “Microscopic Traffic Simulation using SUMO”. In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, pages 2575–2582. DOI: 10.1109/ITSC.2018.8569938 (page 34).

[52] Aparna M P, Gandhiraj R, Manoj Panda. “Steering Angle Prediction for Autonomous Driving using Federated Learning: The Impact of Vehicle-To-Everything Communication”. In: *2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT)*. 2021, pages 1–7. DOI: 10.1109/ICCCNT51525.2021.9580097 (page 28).

[53] Ricardo Maia et al. “Electric vehicle simulator for energy consumption studies in electric mobility systems”. In: *2011 IEEE Forum on Integrated and Sustainable Transportation Systems*. 2011, pages 227–232. DOI: 10.1109/FISTS.2011.5973655 (pages 34, 88).

[54] Akhil Mathur et al. *On-device Federated Learning with Flower*. arXiv:2104.03042 [cs]. April 2021. DOI: 10.48550/arXiv.2104.03042. URL: http://arxiv.org/abs/2104.03042 (visited on July 6, 2022) (page 31).

[55] H. Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: (2016). DOI: 10.48550/ARXIV.1602.05629. URL: https://arxiv.org/abs/1602.05629 (pages 22, 45).

[56] José Ángel Morell, Enrique Alba. "Dynamic and adaptive fault-tolerant asynchronous federated learning using volunteer edge devices". In: *Future Generation Computer Systems* 133 (2022), pages 53–67. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2022.02.024. URL: https://www.sciencedirect.com/science/article/pii/S0167739X22000735 (page 23).

[57] Philipp Moritz et al. *Ray: A Distributed Framework for Emerging AI Applications*. 2017. DOI: 10.48550/ARXIV.1712.05889. URL: https://arxiv.org/abs/1712.05889 (page 50).

[58] Martin Murnane, Adel Ghazel. "A Closer Look at State of Charge (SOC) and State of Health (SOH) Estimation Techniques for Batteries". In: 2017 (page 16).

[59] SUMO Netconvert. *SUMO Netconvert*. URL: https://sumo.dlr.de/docs/netconvert.html (page 41).

[60] SUMO Netedit. *SUMO Netdit*. URL: https://sumo.dlr.de/docs/Netedit/index.html (page 43).

[61] Anh Nguyen et al. *Deep Federated Learning for Autonomous Driving*. 2021. DOI: 10.48550/ARXIV.2110.05754. URL: https://arxiv.org/abs/2110.05754 (page 15).

[62] OpenStreetMap contributors. *Planet dump retrieved from https://planet.osm.org*. https://www.openstreetmap.org. 2017 (page 41).

[63] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pages 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (page 65).

[64] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pages 2825–2830 (page 54).

[65] Jason Posner et al. "Federated Learning in Vehicular Networks: Opportunities and Solutions". In: *IEEE Network* 35.2 (2021), pages 152–159. DOI: 10.1109/MNET.011.2000430 (page 27).

[66] PySyft. *PySyft*. URL: https://github.com/OpenMined/PySyft (page 29).

[67] Ravish Raj. *Regression metrics*. URL: https://www.enjoyalgorithms.com/blog/evaluation-metrics-regression-models (page 54).

[68] Ashish Rauniyar et al. *Federated Learning for Medical Applications: A Taxonomy, Current Trends, Challenges, and Future Research Directions*. 2022. DOI: 10.48550/ARXIV.2208.03392. URL: https://arxiv.org/abs/2208.03392 (page 21).

[69] Sashank Reddi et al. *Adaptive Federated Optimization*. 2020. DOI: 10.48550/ARXIV.2003.00295. URL: https://arxiv.org/abs/2003.00295 (page 22).

[70] Nicola Rieke et al. "The future of digital health with federated learning". In: *npj Digital Medicine* 3.1 (September 2020). DOI: 10.1038/s41746-020-00323-1. URL: https://doi.org/10.1038%2Fs41746-020-00323-1 (page 21).

[71] Nuria Rodríguez-Barroso et al. "Federated Learning and Differential Privacy: Software tools analysis, the Sherpa.ai FL framework and methodological guidelines for preserving data privacy". In: *Information Fusion* 64 (2020), pages 270–292. ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2020.07.009. URL: https://www.sciencedirect.com/science/article/pii/S1566253520303213 (page 29).

[72] Lutz Roeder. *lutzroeder/netron: 5.5.3*. Version v5.5.3. January 2022. DOI: 10.5281/zenodo.5854962. URL: https://doi.org/10.5281/zenodo.5854962 (page 66).

[73] Holger R. Roth et al. "Federated Learning for Breast Density Classification: A Real-World Implementation". In: *Domain Adaptation and Representation Transfer, and Distributed and Collaborative Learning*. Springer International Publishing, 2020, pages 181–191. DOI: 10.1007/978-3-030-60548-3_18. URL: https://doi.org/10.1007%2F978-3-030-60548-3_18 (pages 15, 21).

[74] Holger R. Roth et al. *NVIDIA FLARE: Federated Learning from Simulation to Real-World*. 2022. DOI: 10.48550/ARXIV.2210.13291. URL: https://arxiv.org/abs/2210.13291 (page 30).

[75] Insaf Sagaama et al. "Evaluation of the Energy Consumption Model Performance for Electric Vehicles in SUMO". In: *2019 IEEE/ACM 23rd International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*. 2019, pages 1–8. DOI: 10.1109/DS-RT47707.2019.8958704 (pages 38, 39).

[76] Insaf Sagaama et al. "Proposal of More Accurate Energy Model of Electric Vehicle For SUMO". In: *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)* (2018), pages 464–469 (page 38).

[77] Julio A. Sanguesa et al. "Analyzing the Impact of Roadmap and Vehicle Features on Electric Vehicles Energy Consumption". In: *IEEE Access* 9 (2021), pages 61475–61488. DOI: 10.1109/ACCESS.2021.3072979 (page 39).

[78] Abdul Rahman Sani, Muneeb Ul Hassan, Jinjun Chen. *Privacy Preserving Machine Learning for Electric Vehicles: A Survey*. 2022. DOI: 10.48550/ARXIV.2205.08462. URL: https://arxiv.org/abs/2205.08462 (page 16).

[79] Yuris Mulya Saputra et al. "Federated Learning Meets Contract Theory: Economic-Efficiency Framework for Electric Vehicle Networks". In: *IEEE Transactions on Mobile Computing* 21.8 (2022), pages 2803–2817. DOI: 10.1109/TMC.2020.3045987 (page 27).

[80] Alicante-Murcia Freeway Scenario. *Alicante-Murcia Freeway Scenario*. URL: https://github.com/msepulcre/Alicante-Murcia-SUMO-Scenario (page 34).

[81] Marco Schreyer, Timur Sattarov, Damian Borth. "Federated and Privacy-Preserving Learning of Accounting Data in Financial Statement Audits". In: *ArXiv* abs/2208.12708 (2022) (page 22).

[82] Micah J Sheller et al. *Multi-Institutional Deep Learning Modeling Without Sharing Patient Data: A Feasibility Study on Brain Tumor Segmentation*. 2018. DOI: 10.48550/ARXIV.1810.04304. URL: https://arxiv.org/abs/1810.04304 (page 21).

[83] statista. *Mobility Markets - Electric Vehicles*. URL: https://www.statista.com/outlook/mmo/electric-vehicles/germany (page 16).

[84] SUMO. *SUMO Electric*. URL: https://sumo.dlr.de/docs/Models/Electric.html (pages 40, 43).

[85] SUMO. *SUMO utiltiy duarouter*. URL: https://sumo.dlr.de/docs/duarouter.html (page 42).

[86] SUMO. *SUMO utiltiy randomTrips.py*. URL: https://sumo.dlr.de/docs/Tools/Trip.html (page 42).

[87] Latanya Sweeney. "K-Anonymity: A Model for Protecting Privacy". In: *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.* 10.5 (October 2002), pages 557–570. ISSN: 0218-4885. DOI: 10.1142/S0218488502001648. URL: https://doi.org/10.1142/S0218488502001648 (page 23).

[88] Kang Tan et al. "Federated Machine Learning in Vehicular Networks: A summary of Recent Applications". In: *2020 International Conference on UK-China Emerging Technologies (UCET)*. 2020, pages 1–4. DOI: 10.1109/UCET51115.2020.9205482 (pages 16, 28).

[89] Tensorflow. *TensorFlow Federated: Machine Learning on Decentralized Data*. URL: https://www.tensorflow.org/federated (page 29).

[90] Aso Validi, Walter Morales-Alvarez, Cristina Olaverri-Monreal. "Analysis of the Battery Energy Estimation Model in SUMO Compared with Actual Analysis of Battery Energy Consumption". In: *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*. 2021, pages 1–6. DOI: 10.23919/CISTI52073.2021.9476579 (pages 35, 87).

[91] Aso Validi, Nikita Smirnov, Cristina Olaverri-Monreal. "Metamodel-based simulation to assess platooning on battery energy consumption". In: *Transportation Research Part D: Transport and Environment* 109 (2022), page 103350. ISSN: 1361-9209. DOI: https://doi.org/10.1016/j.trd.2022.103350. URL: https://www.sciencedirect.com/science/article/pii/S136192092200178X (page 35).

[92] Savvas Varsamopoulos, Koen Bertels, Carmen Almudever. "Designing neural network based decoders for surface codes". In: (November 2018) (page 25).

[93] Jianyu Wang et al. *Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization*. 2020. DOI: 10.48550/ARXIV.2007.07481. URL: https://arxiv.org/abs/2007.07481 (page 23).

[94] SUMO Website. *SUMO Website*. URL: https://www.eclipse.org/sumo/ (page 34).

[95] Qi Xia et al. "A survey of federated learning for edge computing: Research problems and solutions". In: *High-Confidence Computing* 1.1 (2021), page 100008. ISSN: 2667-2952. DOI: https://doi.org/10.1016/j.hcc.2021.100008. URL: https://www.sciencedirect.com/science/article/pii/S266729522100009X (page 15).

[96] Chenhao Xu et al. *Asynchronous Federated Learning on Heterogeneous Devices: A Survey*. 2021. DOI: 10.48550/ARXIV.2109.04269. URL: https://arxiv.org/abs/2109.04269 (page 23).

[97] Wansheng Yang et al. "Federated Transfer Learning for Bearing Fault Diagnosis Based on Averaging Shared Layers". In: *2021 Global Reliability and Prognostics and Health Management (PHM-Nanjing)*. 2021, pages 1–7. DOI: 10.1109/PHM-Nanjing52125.2021.9612761 (page 28).

[98] Ashkan Yousefpour et al. "Opacus: User-Friendly Differential Privacy Library in PyTorch". In: *arXiv preprint arXiv:2109.12298* (2021) (pages 23, 51, 52, 86).

[99] Hongyi Zhang, Jan Bosch, Helena Holmström Olsson. *Real-time End-to-End Federated Learning: An Automotive Case Study*. 2021. DOI: 10.48550/ARXIV.2103.11879. URL: https://arxiv.org/abs/2103.11879 (page 28).

[100] Wanru Zhao et al. *Protea: Client Profiling within Federated Systems using Flower*. arXiv:2207.01053 [cs] version: 1. July 2022. URL: http://arxiv.org/abs/2207.01053 (visited on July 25, 2022) (page 31).

[101] Yue Zhao et al. "Federated Learning with Non-IID Data". In: (2018). DOI: 10.48550/ARXIV.1806.00582. URL: https://arxiv.org/abs/1806.00582 (page 23).

[102] Jiehan Zhou et al. *A Survey on Federated Learning and its Applications for Accelerating Industrial Internet of Things*. 2021. DOI: 10.48550/ARXIV.2104.10501. URL: https://arxiv.org/abs/2104.10501 (page 19).

[103] Hangyu Zhu et al. *Federated Learning on Non-IID Data: A Survey*. 2021. DOI: 10.48550/ARXIV.2106.06843. URL: https://arxiv.org/abs/2106.06843 (pages 23, 63, 65).

[104] Alexander Ziller et al. "PySyft: A Library for Easy Federated Learning". In: 2021 (page 29).

All links were last followed on November 9, 2022.

A Appendix

Table A.1: Numeric representation of the MSE, RMSE, MAE and Adjusted R2 for 25 sample clients; Results obtained after 3 FL rounds with 1000 training clients

Client ID	MSE			RMSE			MAE			Adjusted_r2		
	1	2	3	1	2	3	1	2	3	1	2	3
0	0.0042	0	0	0.0645	0.0058	0.0048	0.0643	0.0051	0.0036	-2.2055	0.974	0.9826
6	0.0034	0.0026	0.0024	0.058	0.0511	0.0491	0.0457	0.0447	0.0422	0.3628	0.5068	0.5446
21	0.0056	0	0.0001	0.0751	0.0054	0.0083	0.0746	0.0038	0.0065	0.1951	0.9959	0.9903
33	0.0039	0.0001	0.0001	0.0625	0.0097	0.0071	0.0619	0.0088	0.0062	0.2297	0.9816	0.9901
76	0.004	0.0001	0	0.0636	0.0077	0.005	0.0635	0.0066	0.0043	-0.594	0.9769	0.9903
211	0.0037	0.0002	0.0001	0.0606	0.0126	0.0091	0.0604	0.0119	0.0076	0.3701	0.973	0.9857
251	0.0019	0.001	0.0008	0.0431	0.0313	0.0281	0.0396	0.0282	0.0253	-6.6939	-3.072	-2.272
259	0.0039	0.0003	0.0002	0.0621	0.0171	0.0143	0.0599	0.0158	0.0129	0.2956	0.9462	0.9626
292	0.0037	0.0001	0.0001	0.0605	0.0098	0.0091	0.0599	0.0086	0.0082	-0.6559	0.9565	0.9628
303	0.0035	0.0001	0	0.0593	0.0094	0.0069	0.0592	0.009	0.0059	-0.2629	0.9682	0.9828
389	0.0028	0.0002	0.0001	0.053	0.0146	0.0115	0.0527	0.0144	0.0111	-2.266	0.7507	0.8459
399	0.0035	0.0001	0	0.0592	0.0086	0.0061	0.059	0.0073	0.0053	-2.5149	0.9262	0.9625
415	0.0049	0	0.0001	0.0699	0.0046	0.0085	0.0698	0.0034	0.0082	-15.9296	0.928	0.751
480	0.0039	0.0002	0.0001	0.0622	0.0153	0.0099	0.062	0.0126	0.0071	-0.8494	0.8879	0.9532
527	0.0032	0.0002	0.0001	0.0563	0.0139	0.0097	0.0562	0.0132	0.0086	-0.3534	0.9169	0.9599
648	0.006	0.0002	0.0003	0.0774	0.0148	0.0172	0.0751	0.0115	0.0119	-7.6682	0.6816	0.5697
662	0.003	0.0005	0.0003	0.0551	0.0226	0.0159	0.0549	0.02	0.0128	0.0718	0.8438	0.9226
742	0.0036	0.0001	0	0.06	0.0116	0.0068	0.06	0.0103	0.0058	-0.2499	0.953	0.9839
780	0.0046	0	0	0.0675	0.0043	0.0056	0.0674	0.0035	0.0047	-4.6198	0.9776	0.9617
800	0.0041	0.0001	0	0.0642	0.0104	0.005	0.064	0.0084	0.0033	-4.9301	0.8456	0.9648
858	0.0052	0.0001	0.0001	0.072	0.0075	0.0106	0.0716	0.0055	0.0091	-11.2022	0.8688	0.7377
936	0.0051	0	0.0001	0.0713	0.0058	0.0097	0.0712	0.005	0.0095	-23.1112	0.8388	0.5519
1013	0.0033	0.0003	0.0002	0.0571	0.0166	0.0125	0.057	0.0144	0.01	0.1291	0.9263	0.9583
1024	0.0034	0.0003	0.0002	0.0579	0.018	0.0134	0.0576	0.0147	0.0103	-0.0455	0.8994	0.9443
1057	0.0045	0.0001	0	0.0672	0.01	0.0065	0.0671	0.0089	0.0058	0.464	0.9882	0.995

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature