iFLOW: An Intelligent and Scalable Multi-Model Federated Learning Framework on the Wheels

Qiren Wang, *Student Member, IEEE*, Yongtao Yao, Nejib Ammar, and Weisong Shi, *Fellow, IEEE*

***Abstract*— The high mobility characteristics of connected vehicles present noteworthy difficulties in the domain of federated learning. Based on our understanding, current federated learning strategies do not tackle the challenge of continuously training multiple models for vehicles in constant motion, which are subject to variable network conditions and changing environments. In response to this challenge, we have created and implemented iFLOW, a versatile and intelligent multi-model federated learning infrastructure specifically designed for highly mobile-connected vehicles. iFLOW addresses these challenges by integrating four key aspects: (1) a strategically devised model allocation algorithm that dynamically selects vehicle computing units for distinct model training tasks, optimizing for both resource efficiency and performance; (2) a dynamic client vehicle joining mechanism that ensures smooth participation of vehicles, even in the face of signal loss or weak connectivity, mitigating disruptions in the training process; (3) integration of a large language model (Llama3.3 70B) as an intelligent arbiter for decision-making within the framework, enhancing adaptability and robustness; and (4) real-world deployment and testing on distributed vehicular devices to validate the approach. The experimental evaluation demonstrates that iFLOW allows multiple models to train asynchronously and outperform centralized training. These results affirm the effectiveness of iFLOW in practical, real-world scenarios involving highly mobile vehicular networks.**

***Index Terms*— Multi-model federated learning, connected and autonomous vehicles (CAVs), model scheduling, large language model(LLM).**

I. Introduction

REMARKABLE advancements in communication, robotics, and edge computing have significantly expanded the potential of connected vehicles. They can now interact with one another, as well as with intelligent infrastructures. To enhance safety, reliability, entertainment, and security. The Automotive Edge Computing Consortium (AECC) forecasts that by 2025, every newly manufactured vehicle will possess connectivity features. This would mean that half of all vehicles in circulation nationally will be connected. Moreover, the total count of connected vehicles is projected to hit a staggering 400 million.

Received 12 June 2024; revised 21 November 2024 and 18 March 2025; accepted 22 May 2025. This work was supported in part by U.S. National Science Foundation under Grant 2140346. The Associate Editor for this article was S. H. Ahmed Shah. *(Qiren Wang and Yongtao Yao contributed equally to this work.) (Corresponding author: Qiren Wang.)*

Qiren Wang, Yongtao Yao, and Weisong Shi are with the Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716 USA (e-mail: qirenw@udel.edu).

Nejib Ammar is with InfoTech Lab, Toyota North America, Mountain View, CA 94043 USA.

Digital Object Identifier 10.1109/TITS.2025.3578586

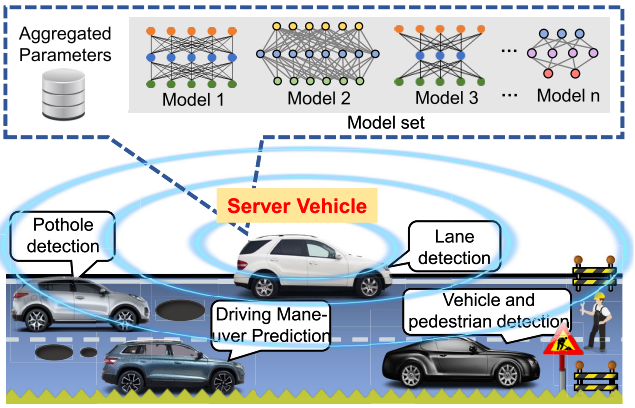


Fig. 1. An illustration of iFLOW.

Simultaneously, a series of deep learning models are being utilized for various connected vehicle applications. These include real-time remote diagnostics [1] and advanced driver assistance [2], which are fueled by the vast and multi-modal vehicle data generated by several onboard sensors such as cameras, radar, and LiDAR. It's projected that upwards of 50 unique deep learning models will operate—frequently simultaneously within a single vehicle. These models will support four main application categories: safety, mobility, information, and computation [3].

A. Federated Learning in Mobility

Training deep learning models on resource-constrained vehicle units is challenging due to increasing model complexity and the demanding nature of the training process [1]. Federated learning provides a solution by enabling distributed vehicles to collaboratively train models locally, sharing only model parameters with the server. This ensures that no raw data is uploaded, enhancing privacy and security. The edge nodes directly collect and store data, which is typically non-independent and non-identically distributed (non-i.i.d.), aligning well with the variability of vehicle sensors and environments [4].

B. Multi-Model Federated Learning

While federated learning has been explored, the concurrent training of multiple independent models on vehicle networks remains underexplored. We introduce iFLOW, a multi-model federated framework designed for distributed vehicle environments. Each vehicle can train one of several models (e.g., for

pothole detection or lane detection), and the server aggregates these updates. iFLOW allows for multiple models to be trained simultaneously across real-world vehicular clusters.

C. Dynamic Management of Client Vehicles

Conventional federated frameworks assume static client nodes, which contradicts the mobility of vehicles. In contrast, iFLOW supports dynamic client management, allowing vehicles to join or leave the cluster as connectivity changes, addressing real-world mobility challenges.

D. Allocating Models to Heterogeneous and Dynamic Vehicle Computing Resources

Vehicle computing capabilities vary widely, with differences in microcontrollers, processors, and accelerators (e.g., GPUs, FPGAs). iFLOW ensures efficient, concurrent model training across heterogeneous vehicle platforms, dynamically allocating models based on available resources.

E. Non-i.i.d Data Distribution Across Different Vehicles

The performance of federated learning can degrade in the presence of non-i.i.d data, especially for algorithms like FedAvg. Vehicles generate unique data due to diverse driving conditions and sensors, complicating convergence. iFLOW addresses this challenge by managing non-i.i.d. data across distributed environments.

Current research on multi-model federated learning is primarily limited to using benchmark datasets and simulations on a single machine. Typically, these studies create multiple virtual nodes on the same device, treating each node as an independent entity. However, such frameworks are often restricted to training a single model at a time and lack actual development and deployment on real devices. These approaches fail to address the network challenges posed by real-world distributed environments, especially in vehicular scenarios. The network is stable if nodes are created on the same machine. All nodes will use the host machine to receive the network signal. To overcome these limitations, we propose the iFLOW framework, which enables concurrent training of multiple models on real distributed devices. The framework considers the real environment of autonomous vehicles. iFLOW effectively addresses the mobility and network constraints inherent in connected vehicle systems, filling a critical gap in existing methodologies.

Our motivation is that in the future, vehicles will not only serve as transportation but also as mobile computing platforms. Autonomous vehicles (AVs) will deploy various models to perform diverse tasks, such as lane detection, object recognition, and driving maneuver predictions [5]. These models will require frequent updates to remain effective in dynamic environments, and federated learning offers a scalable solution for real-time updates while maintaining data privacy. By leveraging federated learning, iFLOW allows autonomous vehicles to continuously update models in real-time, ensuring they can adapt to new conditions and perform tasks efficiently.

In this study, we introduce a multi-model federated learning framework and conduct various design explorations and experiments to tackle the challenges associated with real-world applications. At the same time, we deploy our iFLOW framework on real distributed devices, which are specific for autonomous vehicles such as the NVIDIA Orin NX series, to validate reliability and scalability. The main contributions of this paper are as follows:

1) We propose and deploy a scalable framework for federated learning in connected vehicles. The framework efficiently leverages the heterogeneous and dynamic computing resources within each vehicle for parallel multi-model training, thus optimizing overall training efficiency while ensuring fairness among individual models.
2) To enable dynamic management of client vehicles, we bring ZMQ into our framework to enable joining the cluster for training at any time. This protocol can adapt to vehicle clients with dynamically changing availability. It can address lost or weak network issues that may preclude some vehicles from participating in the cluster.
3) We propose a LLM-based model allocation algorithm to facilitate the simultaneous training of multiple deep learning models across heterogeneous and dynamic vehicle computing resources. This LLM-based algorithm selects an appropriate model for vehicle computing devices, permitting efficient and effective large-scale training.
4) We demonstrate the effectiveness of iFLOW through extensive experimentation with models of varying scales on large-scale datasets. iFLOW offers the advantage of preserving client vehicles so they can train their data without sharing. Moreover, it can significantly improve each model's performance.

The remainder of this paper is structured as follows: Section II reviews related work and the foundational elements of the multi-model federated learning framework. Section III outlines the design of iFLOW and associated methods. Section IV provides extensive experimental results, followed by discussions in Section V. Finally, Section VI offers a conclusion for the paper and discusses future work and current limitations in the last Section.

II. Related Work

Decentralized systems enhance user control over mobility data through privacy-preserving blockchain analytics [6], while edge computing in IoT-driven Intelligent Transportation Systems (ITS) boosts QoS for vehicle-to-vehicle (V2V) communications [7]. Parallel computing accelerates solutions to nonlinear differential systems [8], and latency-aware strategies optimize cloud networks [9]. Additionally, deep learning advancements in autonomous systems and infrastructure monitoring have significantly contributed to smart mobility and intersect with federated learning. For instance, Faster R-CNN has been applied to detect structural damages with 87.8% mean average precision [10], while a 3D pothole segmentation and volume prediction model improves road surface assessment for autonomous driving [11]. This section reviews federated learning progress, including multi-modal methods, client selection, model allocation, and non-i.i.d. data solutions.

A. Applications in Autonomous Systems and Infrastructure Monitoring

While the focus remains on federated learning for mobility services, advancements in deep learning for infrastructure monitoring provide valuable insights. CNN-based methods have achieved high accuracy in detecting concrete cracks in Structural Health Monitoring (SHM) [12], and real-time crack segmentation networks further enhance industrial damage detection efficiency [13]. UAV-based inspection techniques, integrating deep learning and ultrasonic beacons, have improved geo-tagged assessments of bridges and buildings [14]. To address challenges posed by complex backgrounds, hybrid pixel-level segmentation techniques offer robust solutions for infrastructure maintenance [15]. These approaches contribute to safer and more efficient autonomous systems, reinforcing the role of deep learning in mobility networks. On top of that, some well-established deep learning models for object detection like Fast-RCN [16], maneuver intent prediction, which uses LSTM-based methods, and lane detection(e.g., U-Net [17] and LaneNet [18]) make the autonomous vehicles more intelligent. By integrating these advanced models into autonomous systems, safety is significantly enhanced.

B. Federated Learning

1) Conventional Federated Learning: Federated learning was first introduced by researchers at Google. To establish a benchmark for evaluating advancements, Caldas et al. developed LEAF [19], a framework designed for federated learning scenarios. LEAF provides six open-source datasets and a conventional setup where a global model on the server aggregates updates from local client models. However, it lacks support for distributed execution, limiting its applicability in real-world federated settings with heterogeneous devices. While LEAF offers diverse datasets for evaluating federated learning approaches, none specifically focus on mobility services, highlighting a gap in benchmarking resources for this domain. Nonetheless, it remains a foundational tool for advancing federated learning research across various applications.

C. Multi-Model Federated Learning

The concept of concurrently training multiple independent deep learning models within a federated framework is relatively novel. Only a handful of studies have proposed algorithms related to multi-model federated learning, and none of these have considered their application in connected mobility.

For instance, Bhuyand et al. expanded the LEAF framework to accommodate multi-model federated learning and introduced two novel client selection strategies [4]. In a unified multi-model training context, decisions are made predicated on the local loss of each client-model pair. The performance of the proposed multi-model strategies demonstrated no inferiority to single-model training utilizing FedAvg. However, this research only considered two models, with a single model allocated to each client.

TABLE I
COMPARISON OF iFLOW WITH EXISTING APPROACHES

Feature	iFLOW	LEAF [19]	Bhuyan [4]	Li [20]
Real-Device Deployment	✓	✓	×	×
Multi-Model FL	✓	×	✓	✓
Dynamic Aggregation	✓	×	×	×
LLM for Model Allocation	✓	×	×	×

Later on, Bhuyand et al. introduced two variants of the well-known FedAvg algorithm for multi-model federated learning, namely, multi-federal incremental random (MFA-Rand) and multi-federal savings rotation mechanism (MFA-RR). These variants employ a common set of clients to train multiple models concurrently, and experimental results illustrate that this approach can yield superior performance compared to training each model individually for the same computational resources.

Moreover, Li et al. cast multi-model training as an optimization problem and developed a logarithmic fairness-based multi-model balancing algorithm (LFMB) [20]. This algorithm cyclically goes through the already assigned models with an unassigned model at each client, enhancing training efficiency until no further improvement is detectable. Experimental evidence highlights the notable superior performance of LFMB in terms of overall training efficiency and model fairness.

Furthermore, Muhammad et al. proposed a robust multi-model federated learning (RMMFL) framework. This framework incorporates a high-entropy aggregation method to soften output predictions and utilizes a weighted integration technique to assess the predictions of each client model based on the performance of each client model. Their simulation outcomes indicate that RMMFL outperforms the baseline approach by an accuracy increase of 5%, thereby significantly improving the learning outcomes of each individual model.

Additionally, Kowsari et al. presented a novel concept in multi-model federated learning. They proposed a random multi-model deep learning for classification (RMDL) where multiple models can be constructed to analyze the same dataset, and their outputs can be aggregated to obtain highly precise results [21]. RMDL comprises three random models: one DNN classifier on the left, one Deep CNN classifier in the middle, and one Deep RNN classifier on the right.

Table I highlights the significant research gaps addressed by our work compared to previous federated learning frameworks.

D. Client Selection

Several client selection algorithms have been developed for single-task federated learning. Cho et al. provide the first convergence analysis of joint optimization for biased client selection, assessing its impact on convergence speed [22]. They introduce POWER-OF-CHOICE, an adaptive policy that balances convergence speed and bias, achieving three times faster convergence and a 10% higher test accuracy than random joint averaging. However, its fairness and robustness could be improved. Later, Cho et al. propose UCB-CS, a bandit-based, communication-efficient strategy that enhances convergence speed and fairness compared to baselines, though

it assumes independent local losses, which may not hold in practice [23]. Latency from numerous communication rounds remains a challenge.

To address training latency, Xia et al. present an online client scheduling (CS) framework using a multi-armed bandit approach, independent of wireless channel or client statistics [24]. They propose CS-UCB for ideal i.i.d. and balanced datasets, and CS-UCB-Q, incorporating virtual queueing, for non-i.i.d., unbalanced datasets with varying client availability [25], [26].

E. Model Allocation and Scheduling

1) Task-Level Scheduling on Heterogeneous Platforms: Prior to the broad utilization of Deep Neural Network (DNN) models, task-level scheduling on heterogeneous platforms was a topic of substantial research. For instance, Augonnet et al. [27] developed Starpu, a framework designed to enhance the execution efficiency of specific conventional algorithms on CPU and GPU platforms. Li et al. [28] turned their attention to the energy consumption issue of heterogeneous computing systems (HCS), exploring solutions to optimize power usage. In a similar way, Chronaki et al. proposed OmpSs, a solution aimed at boosting the performance of parallel computing in heterogeneous multi-core systems. Furthermore, AlEbrahim and Ahmad [29] introduced a scheduling algorithm that calculates the priority of each task and designates a processor to handle each task. To facilitate low-latency prediction serving, Crankshaw et al. [30] proposed Clipper, a system that enables simplified model deployment across various frameworks and applications. At the same time, Zhang et al. [31] developed Mark, a general-purpose inference serving system built on Amazon Web Services (AWS), designed to fulfill the response-time Service-Level Objectives (SLOs) of inference workloads while minimizing serving cost. In a similar context, Kannan et al. [32] introduced GrandSLAm, a microservice execution framework designed to optimize utilization in data centers hosting microservices. Lastly, Romero et al. [33] proposed INFaaS, a model-less system for distributed inference serving, where developers merely need to specify the performance requirements of their applications.

F. Approaches to Deal With Non-i.i.d Data

In contrast to centralized training, federated learning typically involves training on non-identically and independently distributed (non-i.i.d.) datasets. To mitigate biases associated with global model updates, Zhao et al. [34] proposed sending a minor fraction of uniformly distributed data to participating clients as a solution to the significant issue posed by non-i.i.d data. However, such an approach inevitably leads to increased communication latency and computational load. In a similar vein, Mohri et al. [35] introduced an agnostic federated learning framework with the aim of ensuring fairness among clients and thus mitigating bias caused by non-i.i.d. data from a variety of clients. Later on, Wang et al. [36] developed FAVOR, a control framework informed by empirical evidence, which intelligently selects the client devices participating in each round of federated learning. This approach was designed to counteract biases introduced by non-i.i.d. data, thereby accelerating the convergence of the model.

G. Large Language Models Integrated for Decision-Making Tasks

Large language models (LLMs) have demonstrated their effectiveness in decision-making tasks. Research has shown that with well-structured prompts, pre-trained LLMs can decompose high-level tasks into intermediate steps without additional training [37], [38]. Their ability to structure complex decision processes has been leveraged in various applications. For instance, LLMs have been used to assist low-level reinforcement learning (RL) agents by providing structured plans and sub-task guidance, ensuring feasibility through pre-trained skill constraints [39]. Additionally, ReAct, a method integrating reasoning and interactive decision-making, enhances LLM performance in tasks such as question-answering and fact verification by generating reasoning traces and task-specific actions [40]. Llama3.3, Meta' latest open-source model, extends these capabilities with improved contextual understanding and efficiency, making it particularly suitable for federated learning and real-time decision-making in resource-constrained environments. Its enhanced architecture enables more effective adaptive learning strategies, reinforcing its role as an intelligent arbiter for optimizing model selection and deployment in dynamic settings.

III. Methodology

The multi-model federated learning approach has particular relevance to the automotive industry due to its ability to amalgamate multiple data sources and models, thereby enhancing the precision and dependability of a vehicle's decision-making system. In this section, we illustrate the design process of our multi-model federated learning framework and delve into the implementation of associated methodologies.

A. Intelligent Multi-Model Federated Learning Framework

1) Framework Details: We envision a scenario where a server vehicle is tasked with training m independent deep learning models in a distributed fashion across a cluster of client vehicles. Each client vehicle holds its own private dataset, which is used to train different models on each car. The server vehicle manages a global version of each of the m models. During each round of the training process, the server vehicle is charged with distributing the parameters of a single model to all client vehicles for the subsequent round of model training. Each time the server receives the corresponding model, profiling information and training results are sent to an intelligent large language model for making decisions for the next round in a specific client. Here, we call this assistance from a large language model as an intelligent arbiter. After the decision comes from an intelligent arbiter, a model that will be trained in the next round will be sent to the client. The client will repeat the training procedure on its local for that round. This process is iteratively repeated as required when the system breaks.

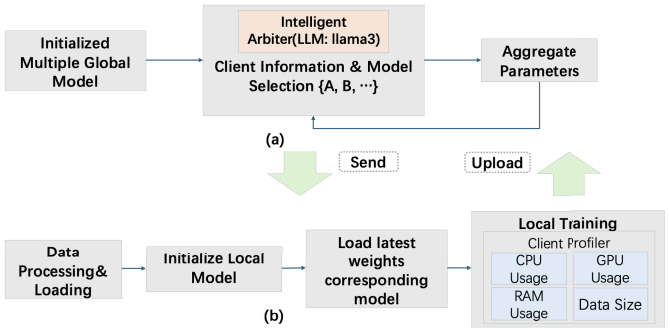


Fig. 2. (a) The workflow of Server Vehicle. (b) The workflow of Client Vehicle. Our iFLOW consists of these two parts. This figure shows how this system works.

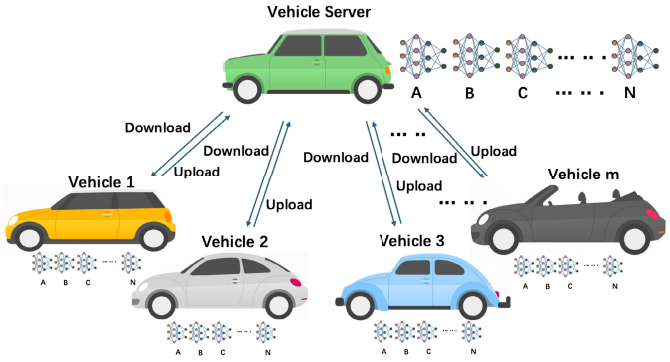


Fig. 3. Overview of Multi-model aggregation.

At the onset of each round, the server vehicle opts for up to m client vehicles for training, where m is a fixed parameter supplied to iFLOW. Each client vehicle receives the global weights of the model it is assigned to train. These clients then train their allocated global model versions using their respective local training datasets and subsequently forward the updated model weights to the server vehicle. The server weights are calculated as an average of the corresponding client weights. FedAvg is doing the weighted averaging process, and we denote α here in the equation:

$$\boldsymbol{w}_{i+1} = (1-\alpha)\cdot \boldsymbol{w}_i + \alpha \cdot \boldsymbol{w}_{i,c}^{client}$$

where $\boldsymbol{w}_i, \boldsymbol{w}_{i,c}^{client}, \boldsymbol{w}_{i+1} \in \mathbb{R}^d$ represent the global model parameters in the i-th communication round, the model parameters received from the c-th client, and the updated global model parameters, respectively. They are all d-dimensional real-valued vectors. α is a hyperparameter in the interval $(0, 1)$ that controls the weight of the client parameters when updating the global model parameters. This method is the standard procedure and is prevalently employed in federated learning [1], [23].

Fig. 2 shows the different workflows between the server vehicle and the client vehicle. Fig. 3 provides a succinct overview of iFLOW, illustrating the collaboration between a server vehicle and a client vehicle. From the figure, the server vehicle performs a mechanism of parameter aggregation. The client vehicle will pick one of the models to train on its

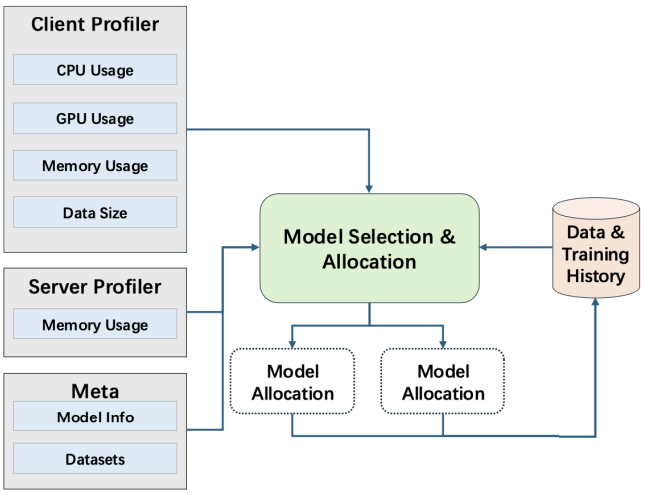


Fig. 4. Metrics for model allocation.

local device and send training weights to the server vehicle. The process initiates with the server vehicle initializing a global model, where all model weights and biases are set to their initial values. All clients are going to train one of the models randomly at the beginning with the default setting. Subsequently, our model allocation mechanism is implemented to delegate the current model training tasks to the client vehicles. It's noteworthy that model allocation is predicated on the initialized model information and the client profiling information (including aspects like CPU usage, GPU usage, memory usage, and dataset size).

Following the description of the procedure on the server, which refers to Algorithm 1, the server vehicle begins by distributing the initialized model parameters to the designated client vehicles. Each client vehicle assumes the critical role of continuously training the models using its local dataset. The client vehicles periodically push their current model parameter values to the server vehicle, which then performs aggregation using the FedAvg aggregation protocol. This aggregation step is crucial for integrating the knowledge obtained from the distributed training process. Once the aggregation is completed, each client vehicle can immediately retrieve the updated parameter values from the server vehicle. These updated parameters are assigned as the current parameters for the respective client vehicles, enabling them to continue the training process with the refined model. This iterative cycle of local training, parameter pushing, aggregation, and parameter pulling allows for the continuous improvement of the global model while leveraging the diverse data available across the fleet of vehicles. Throughout the entire process, the server vehicle plays a vital role in coordinating the communication and aggregation of model parameters. By utilizing the ZMQ protocol, the server vehicle ensures efficient and reliable communication channels with the client vehicles.

2) Metrics for Model Allocation: As illustrated in Algorithm 2, each client vehicle is initialized with a unique identifier (clientID) and establishes a connection to the ZMQ broker using the provided broker address, port, and topic for

Algorithm 1 Pseudo-Code for Multi-Model Aggregation at Server Vehicle

Require: *broker*, *port*, *topicPush*, *topicPull*
Ensure: $globalModel_n \leftarrow \mathbf{0}$, $\forall n \in \{A, B, \ldots, Z\}$; $\alpha \in (0, 1)$; $clientPerfData_c \leftarrow \{\}$, $\forall c \in \{1, 2, \ldots, M\}$
setupZMQ(*broker*, *port*, *topicPush*)
repeat
  onMessage(*client*, *userdata*, *msg*)
  modelName, *clientId*, *clientParams* ← *parseMessage*(*msg*)
  localModel ← *globalModel*[*modelName*]
  clientParams, *perfData* ← *processData*(*clientParams*, *clientId*)
  localModel ← *fedAvg*(*localModel*, *clientParams*, α)
  saveWeights(*localModel*, *modelName*)
  nextModel ← *decideNextModel*(*clientId*, *modelName*)
  publishParams(*nextModel*, *globalModel*[*nextModel*], *clientId*)
until *serverStopped*()
function *publishParams*(*modelName*, *globalModel*, *clientId*)
  params ← *getParams*(*modelName*, *globalModel*)
  message ← *constructMessage*(*modelName*, *params*)
  publish(*message*, *clientId*)
end function
function *decideNextModel*(*clientId*, *currentModel*)
  prompt ← *constructPrompt*(*clientId*, *currentModel*)
  ollamaDecision ← *queryOllama*(*prompt*)
if *isValidDecision*(*ollamaDecision*) **then**
  return *ollamaDecision*
else
  return *randomDecision*()
end if
end function

Algorithm 2 Pseudo-Code for Model Training at Client Vehicle

Require: *clientId*, *broker*, *port*, *topicPush*, *topicPull*
Ensure: *device*, *models*[n], $\forall n \in \{A, \ldots, Z\}$; *modelNextTraining*, *client*, *epoch*, *dataLoader*
setupZMQ(*broker*, *port*, *topicPullClientId*)
repeat
  onMessage(*client*, *userdata*, *msg*):
    modelNextTraining, *params* ← *parseMessage*(*msg*)
    updateModelParameters(*models*[*modelNextTraining*], *params*)
  startTrainModel(*dataLoader*, *epochs*):
    model ← *models*[*modelNextTraining*]
    optimizer, *criterion* ← *setupOptimizerAndCriterion*(*model*)
  for *epoch* **in** *range*(*epochs*) **do**
    for *data*, *target* **in** *dataLoader* **do**
      trainStep(*model*, *optimizer*, *criterion*, *data*, *target*)
    end for
    collectPerformanceMetrics()
  end for
  performanceData ← *calculateAverageMetrics*()
  printResults(*modelNextTraining*, *clientId*, *epoch*, *performanceData*)
  pushParams(*modelNextTraining*, *model*, *performanceData*)
until *clientStopped*()
function *trainStep*(*model*, *optimizer*, *criterion*, *data*, *target*)
  performTrainingStep(*model*, *optimizer*, *criterion*, *data*, *target*)
end function
function *pushParams*(*modelName*, *model*, *performanceData*)
  params ← *getModelParameters*(*model*)
  message ← *constructMessage*(*clientId*, *modelName*, *params*, *performanceData*)
  publish(*message*, *topicPush*)
end function

receiving messages (topic_pull_clientID). The client vehicle awaits incoming messages from the server vehicle, which contain the model to be trained next (modelNextTraining) and the associated parameters. Upon receiving a message, the client vehicle parses the message to extract the modelNextTraining and the corresponding parameters. It then updates the parameters of the specified model in its local collection of models. Subsequently, the client vehicle initiates the training process using the local data loader and the specified number of epochs. During the training process, the client vehicle sets up the optimizer and criterion for the model and iteratively performs training steps for each data batch in the data loader. After each epoch, the client vehicle collects performance metrics and calculates the average metrics across all epochs. These training results, including the modelNextTraining, clientID, epoch, and performance data, are printed for monitoring purposes. Finally, the client vehicle pushes the updated model parameters and the collected performance data back to the server vehicle using the pushParams function. This function constructs a message containing the clientID, modelName, params, and performanceData, and publishes it to the designated topic (topic_push) for the server vehicle to receive and process. This process continues iteratively until the client is stopped, allowing for continuous collaborative learning and model improvement across the fleet of vehicles.

For the purpose of model allocation, we integrate the Python *psutil* package to create profiles that collect key metrics associated with the status of client vehicles (CPU usage, GPU usage, memory usage, Dataset size), meta information (like model information and the dataset of each client vehicle). The *psutil* library in Python is constructed atop several Linux system calls to glean system-level information. Specifically, *psutil* utilizes the */proc* file system, which offers an interface for accessing kernel data structures and system details. By reading the files located under */proc*, *psutil* can access a wealth of system information, encompassing CPU, memory, disk and network usage, process details, and more. All these metrics and model allocation outcomes from the previous round are stored in the server vehicle's database as historical data. This historical data

not only provides a comprehensive record but also aids in the decision-making process for model allocation in subsequent rounds.

B. Aggregation Protocols With Intelligent Arbiter

1) Four Aggregation Protocols: In each communication round, the client vehicle acquires the current global model from the server vehicle. Subsequently, client vehicles do training locally utilizing their local data and send updates to the global model in each iteration. Following this, the central server merges these updated parameters to generate the updated global model. This process is known as the aggregation protocol. In this study, we evaluate *FedAvg* among aggregation methods: Federated Averaging (FedAvg), Robust Federated Aggregation (RFA), Clustered Federated Learning (CFL), and Multi-Krum (MKrum). These methods are compared to understand their influence on the performance of the global model's training [41]. Detailed descriptions of these aggregation algorithms are presented in the following sections.

- **Federated averaging (FedAvg)**: This is a conventional aggregation protocol for standard federated learning. FedAvg accepts global parameters from all client vehicles and calculates the weighted average of the latest parameters for updating [42].
- **Robust federated aggregation (RFA)**: This robust aggregation protocol is designed to minimize the weighted geometric median (GM) of global parameters received from all client vehicles. It computes the approximate GM using the smoothed Weiszfeld algorithm [43].
- **Clustered federated learning (CFL)**: This protocol segregates the client vehicle population into two distinct clusters: benign client vehicles and corrupt client vehicles. This division is based on the pairwise cosine similarity between their latest parameter updates [44].
- **Multi-Krum (MKrum)**: This is a standard Byzantine-tolerant aggregation protocol that can withstand certain Byzantine faults. Such faults could include completely arbitrary behavior of client updates [45].

2) Intelligent Arbiter: In our multi-model federated learning framework, we employ Llama3.3, the latest open-source large language model from Meta, as an intelligent arbiter. Deployed on the server vehicle, Llama3.3 analyzes profiling information from client vehicles, including CPU usage, GPU usage, memory usage, dataset size, and initialized model details. This profiling information is sent in JSON format after each training round to facilitate smooth interaction with the large language model. To determine the next model for training, we designed a query prompt. The prompt combines client identifiers, current model names, performance history, and model structure descriptions to instruct Llama3.3 in selecting the next model based on predefined priorities: (1) balancing training counts across models, (2) avoiding consecutive training of the same model, and (3) considering model structure. The intelligent arbiter's role is crucial in optimizing model selection dynamically, ensuring efficient resource utilization and balanced model performance. This query-based decision-making process is integrated into our experimental design and evaluation. Specifically, we assessed its impact by comparing the training distribution and performance improvements under different model selection strategies, demonstrating the arbiter's effectiveness in real-world deployments.

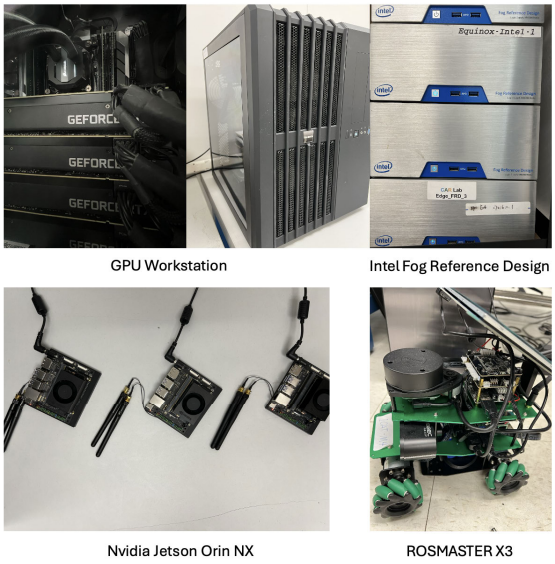


Fig. 5. An illustration of the heterogeneous computing platform.

IV. Experimental Platform

To tackle heterogeneity in vehicular computing, we built a testbed to assess our proposed system. As shown in Fig. 5, it includes a router, a desktop workstation, an Intel Fog Reference Design (Atom E3900, 32GB memory), two GPU workstations (one with four NVIDIA RTX 8000 GPUs and NVLinks for Llama3.3), and three Nvidia Jetson Orin NX boards (16GB memory, Ampere GPU, Arm Cortex CPU) for edge computing in autonomous driving.

Network stability is maintained via a router acting as a wireless access point, with one GPU workstation on Ethernet and others on 2.4 GHz Wi-Fi. A large language model runs on a GPU workstation, the Intel Fog unit acts as the main server, and the Jetson boards handle distributed computing and training.

We also added the ROSMASTER X3 ROS Robot (Fig. 6) for vehicle clustering tests. This ROS-based robot, with Mecanum wheels, LiDAR, depth camera, and voice module, supports multiple Jetson platforms and Python programming for SLAM, tracking, navigation, and more. It offers flexible control via apps, ROS interfaces, or hardware controllers.

V. Experiments

In this section, we first elaborate on the datasets we used for experiments and then describe our model implementation details. Next, we show our hardware profiling evaluation among different boards. Finally, we validate our iFLOW framework's effectiveness and feasibility.

A. Dataset Description

The BDD100K dataset [46] is a large-scale and diverse driving dataset designed for heterogeneous multitask learning

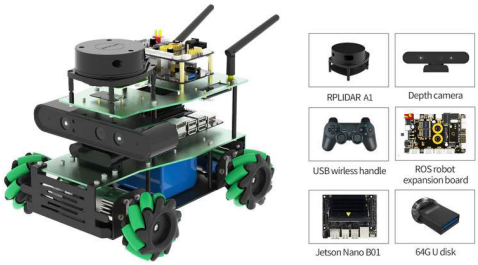


Fig. 6. A ROSMASTER X3 ROS robot.

in autonomous driving. Comprising 100,000 high-resolution videos and images captured from various urban and highway environments, BDD100K provides annotations for a wide range of perception tasks, including object detection, lane detection, drivable area segmentation, and instance segmentation. Specifically, for object detection, the dataset contains over 1.8 million labeled instances across pedestrian and vehicle categories, ensuring robust recognition under diverse lighting and weather conditions.

The Brain4Cars dataset [47] is a large-scale temporal dataset designed for driving maneuver anticipation. It consists of multimodal sensor recordings from real-world driving scenarios. It captures vehicle telemetry, driver gaze, and road context to predict maneuver intentions such as left/right turns, lane changes, and stopping. The dataset is particularly notable for its inclusion of long-term temporal dependencies, making it an essential benchmark for predictive modeling in intelligent driving assistance systems.

The TuSimple dataset is a high-quality public dataset designed specifically for lane detection in highway environments. It comprises video sequences captured from a front-facing camera mounted on autonomous test vehicles, featuring complex lane structures, occlusions, and varying road textures. The dataset provides pixel-wise annotations for lane markings, making it a valuable resource for developing and evaluating robust lane detection algorithms in autonomous driving applications.

Each of these datasets presents unique challenges and opportunities for advancing scene understanding, maneuver intention prediction, and lane detection, which are critical components of intelligent driving systems. Their large-scale and diverse nature makes them indispensable benchmarks for evaluating modern computer vision and machine learning models in autonomous driving. In order to compare iFLOW with other frameworks fairly, we adopt the same dataset for each client while they train the same model.

B. Model Implementation Details

To comprehensively evaluate the proposed intelligent multi-model federated learning framework (iFLOW) and assess its adaptability across diverse autonomous driving scenarios, we implemented three distinct neural network architectures specifically tailored for various perception tasks in autonomous driving. In our federated learning setup, clients locally train their respective models on heterogeneous datasets, periodically synchronizing model parameters with a central server vehicle that aggregates these updates using the Federated Averaging (FedAvg) method.

Model A is an LSTM-based neural network specifically developed for predicting driving maneuvers using sequential vehicle telemetry data from the Brain4Cars dataset. This model features an input dimension of two, capturing the temporal sequences of sensor data, and employs a hidden dimension of 45 units. The network includes an LSTM layer with a dropout rate of 0.3, followed by a dropout layer with a rate of 0.4 to mitigate overfitting. The final classification layer outputs probabilities over five distinct maneuver prediction categories.

Model B targets object detection tasks based on the Faster R-CNN architecture combined with MobileNetV3 and a Feature Pyramid Network (FPN) backbone, specifically tailored for the BDD100K dataset. It starts with an initial convolutional layer having 32 channels, integrated with batch normalization and ReLU activation. The model utilizes multiple bottleneck layers composed of depthwise separable convolutions, facilitating efficient computation without sacrificing accuracy. The FPN aids in robust multi-scale feature extraction essential for accurately identifying pedestrians and vehicles in complex urban and highway driving conditions.

Model C is based on a specialized U-Net architecture aimed at lane detection tasks, optimized for the TuSimple dataset. It consists of multiple double convolutional blocks, each including pairs of convolution layers with kernel sizes of 3 and padding of 1, combined with batch normalization and ReLU activation. Max pooling layers reduce the spatial dimensions before upsampling through transpose convolutions. Each convolutional layer is initialized using Kaiming normal initialization to promote efficient training and rapid convergence. The final layer outputs a segmentation map through a sigmoid activation, enabling precise delineation of lane boundaries even in challenging scenarios.

C. Training Loss and Effectiveness Evaluation

To comprehensively evaluate the effectiveness and efficiency of the proposed iFLOW multi-model federated learning framework, we implemented three neural network models designed for different autonomous driving tasks. As shown in Fig.7, we compare our iFLOW multi-model federated learning framework with RankList-Multi-UCB and Pareto-Multi-UCB [4], which is the state-of-the-art work for multi-model federated learning, across three clients and recorded the training loss trajectories over time. The observed loss trends confirm that all three models exhibit clear convergence patterns across different client environments, validating the effectiveness of the multi-model federated learning approach. Model A, based on an LSTM sequence modeling architecture with a smaller parameter space, initially presented higher loss values but quickly stabilized as training progressed. Models B and C, based on Faster R-CNN and U-Net architectures, respectively, started with relatively lower loss values and converged efficiently to lower stable loss levels.

We further conducted a quantitative analysis of the time required for each framework to reach predefined reference loss values, which were set individually for each client-model combination to assess training efficiency. Table II shows the

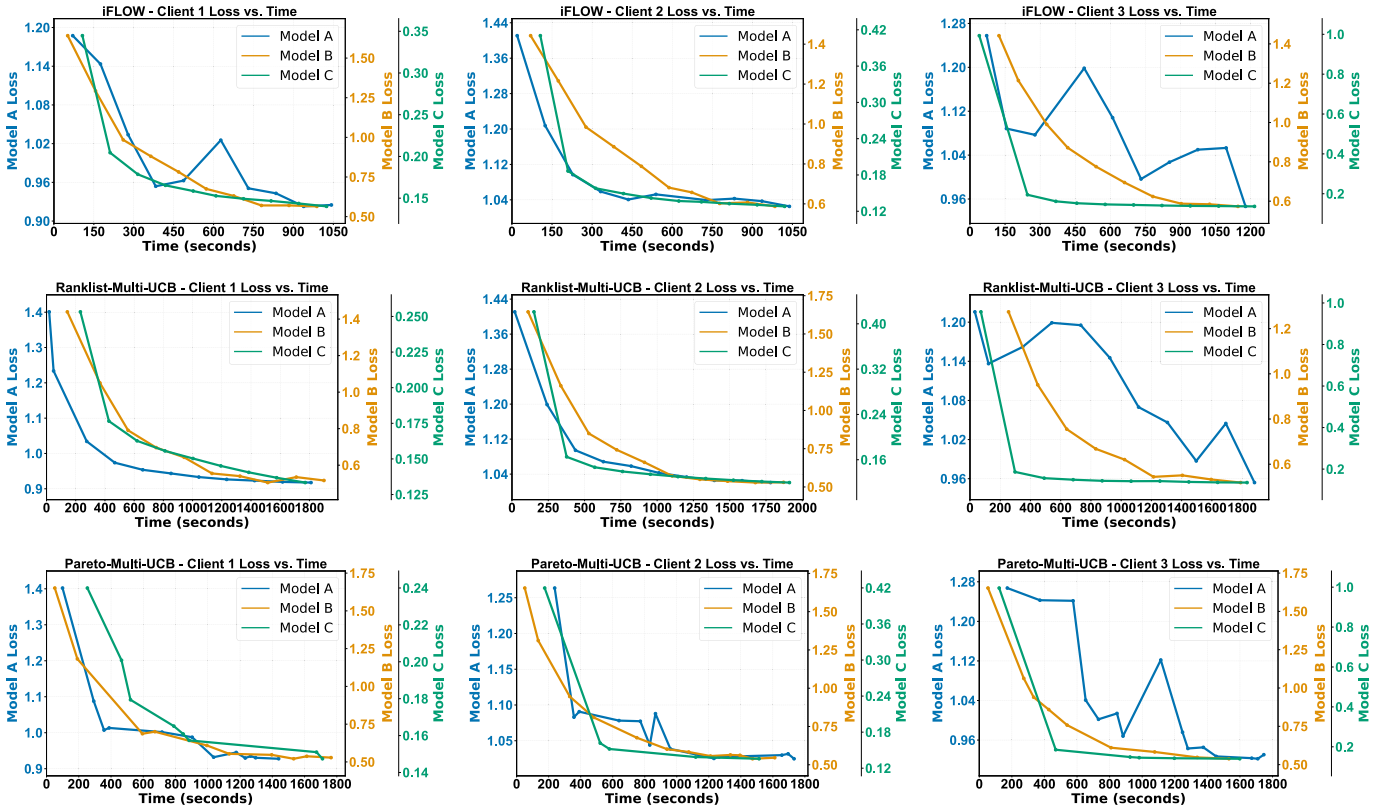


Fig. 7. Training loss vs. Time for model A, B, C on each client.

TABLE II
COMPARISON OF TRAINING EFFICIENCY ACROSS FRAMEWORKS

Client	Model	Time to Target Loss (s)			Epochs		
		iFLOW	Rank	Pareto	iFLOW	Rank	Pareto
Client 1	A (0.95)	**148.1**	211.9	287.8	10	11	12
	B (0.6)	**395.8**	547.2	483.1	10	10	10
	C (0.15)	**254.4**	394.0	460.0	10	9	8
Client 2	A (1.1)	**55.9**	170.4	88.8	10	10	13
	B (0.6)	**490.7**	553.6	617.3	10	10	12
	C (0.15)	**145.4**	120.8	181.3	10	10	5
Client 3	A (0.98)	**319.1**	534.9	354.1	10	11	15
	B (0.6)	**407.8**	626.8	511.4	10	9	9
	C (0.15)	**195.0**	168.2	154.7	10	10	6

summary of training efficiency across all clients and models of iFLOW, RankList-Multi-UCB and Pareto-Multi-UCB. iFLOW outperformed other frameworks consistently and allow each model training fairly for 10 epochs on each client during the 30 training epochs in total. For instance, Client 1's Model A reached its reference loss of 0.95 within 148.1 seconds under iFLOW, compared to 211.9 seconds for RankList-Multi-UCB and 287.8 seconds for Pareto-Multi-UCB, reflecting a speed improvement of approximately 30% and 48%, respectively. Similarly, Model B in Client 1 reached its reference loss of 0.6 in 395.8 seconds under iFLOW, whereas RankList and Pareto required 547.2 seconds and 483.1 seconds, respectively. Model C in Client 1 achieved its reference loss of 0.15 in 254.4 seconds, significantly faster than RankList at 394.0 seconds and Pareto at 460.0 seconds.

The analysis of Clients 2 and 3 further confirmed iFLOW's efficiency advantages. Model A in Client 2 reached its reference loss of 1.1 in 55.9 seconds under iFLOW, nearly three times faster than RankList at 170.4 seconds and substantially better than Pareto at 88.8 seconds. For Model B, iFLOW reached its reference loss of 0.6 in 490.7 seconds, again outperforming RankList at 553.6 seconds and Pareto at 617.3 seconds. In Client 3, Model A achieved its reference loss of 0.98 in 319.1 seconds under iFLOW, significantly lower than RankList at 534.9 seconds and Pareto at 354.1 seconds. For Model B, the reference loss of 0.6 was achieved in 407.8 seconds under iFLOW, compared to 626.8 seconds for RankList and 511.4 seconds for Pareto. Model C in Client 3 reached its reference loss of 0.15 in 195.0 seconds, a performance similar to RankList at 168.2 seconds and Pareto at 154.7 seconds. Despite the close results for Model C, iFLOW demonstrated a more compact and efficient overall training process.

Scheduling efficiency is also illustrated in Fig. 8. The RankList-Multi-UCB and Pareto-Multi-UCB frameworks adopt a synchronous federated learning structure, requiring clients to wait for synchronization, which results in significant idle times and longer overall training durations. In contrast, iFLOW operates asynchronously, allowing each client to immediately begin training the next model upon completing the previous one, significantly reducing idle time between training sessions. For example, the total training duration for Client 1 under iFLOW was 1042.7 seconds, whereas Pareto and RankList required 1764.2 seconds and 1899.5 seconds,

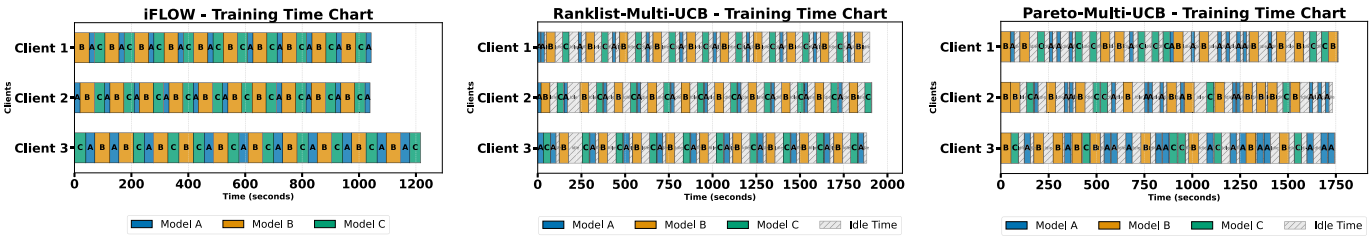


Fig. 8. Scheduling charts (iFLOW, Ranklist-Multi-UCB, Pareto-Multi-UCB) for each clients in different frameworks.

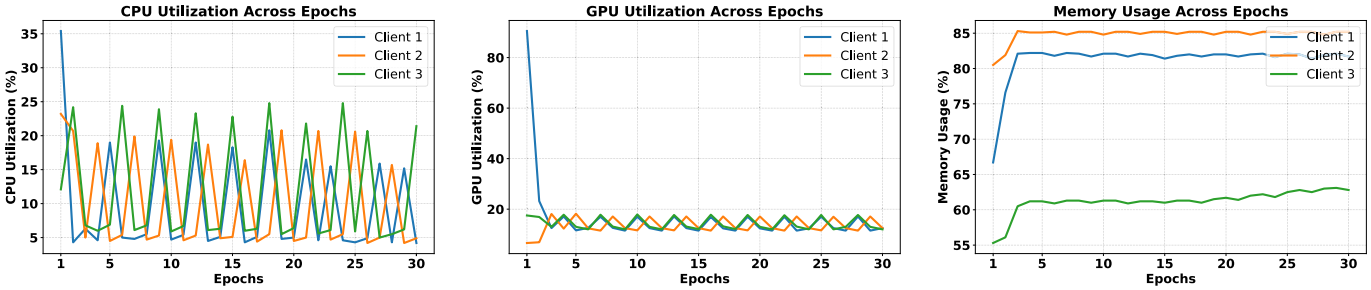


Fig. 9. CPU, GPU, and Memory usage monitoring for Client1, Client2, and Client3 on iFLOW.

respectively, with Pareto incurring 759.7 seconds of idle time and RankList experiencing 876.4 seconds of waiting time. Similar efficiency advantages were observed in Client 2, where iFLOW completed training in 1038.7 seconds, compared to 1732.4 seconds for Pareto and 1911.3 seconds for RankList. In Client 3, iFLOW completed training in 1216.9 seconds, whereas Pareto required 1746.4 seconds and RankList took 1881.9 seconds.

These quantitative results and visualized comparisons strongly validate the effectiveness and efficiency of the proposed iFLOW framework. Through its intelligent asynchronous multi-model federated learning framework, iFLOW significantly improves training efficiency in dynamic and heterogeneous vehicular computing environments, demonstrating its strong potential for practical deployment in autonomous driving applications.

D. Profiling and Hardware Evaluation

To evaluate the efficiency and practicality of our framework, we conducted profiling experiments that measured CPU usage, GPU utilization, memory consumption, and epoch durations for three client vehicles across 30 training epochs, irrespective of the model selected for each epoch. As demonstrated in the subsequent figures, the profiling information (CPU usage, GPU usage, Memory usage) provides insights into the system's resource utilization.

The profiling results indicate distinctive resource usage patterns for each client. Client 1 exhibited an average CPU usage ranging from approximately 4% to 35%, with GPU utilization initially peaking at about 90.5%, then stabilizing between 11.5% and 17.1%. Client 2 consistently showed high memory usage at around 85%, accompanied by GPU utilization varying between 11.6% and 18.2%. Conversely, Client 3 demonstrated lower memory utilization, staying between 61% and 63%, with GPU usage ranging from 12% to 18%, similar to Client 2. Epoch durations varied significantly among the clients, ranging from approximately 20 to 78 seconds, highlighting the resource heterogeneity inherent in real-world distributed environments. The detailed results for CPU, GPU, and memory utilization across epochs are illustrated in Fig. 9. These profiling outcomes confirm that our framework effectively accommodates diverse computational capabilities within real-world distributed scenarios.

VI. Conclusion

In this paper, we introduced iFLOW, a scalable multi-model federated learning framework for connected vehicles. Unlike prior approaches that rely on single-machine simulations, iFLOW supports real-world, asynchronous training across distributed vehicular devices, improving adaptability and efficiency in dynamic environments. The integration of Llama3.3 enhances adaptive model allocation, optimizing resource utilization while maintaining robustness and fairness.

Experiments demonstrate that iFLOW significantly improves training efficiency and model performance by enabling concurrent multi-model training, reducing synchronization delays, and handling heterogeneous computing power effectively.

Despite these strengths, some challenges remain. Frequent disconnections due to vehicle mobility may disrupt training, requiring mobility-aware scheduling and recovery mechanisms. Inter-vehicle distances, if too large, can impact model aggregation, necessitating better clustering and communication strategies. Heterogeneous computing capabilities can lead to training speed variations, affecting global model convergence, which can be mitigated by dynamic scheduling. Additionally, large-scale model transmission poses bandwidth challenges, highlighting the need for compression and incremental updates.

To advance research in this field, we will open-source iFLOW on GitHub, facilitating further innovation in

multi-model federated learning for connected and autonomous vehicles.

REFERENCES

[1] S. Lu, Y. Yao, and W. Shi, "CLONE: Collaborative learning on the edges," *IEEE Internet Things J.*, vol. 8, no. 13, pp. 10222–10236, Jul. 2021.

[2] P. M. Greenwood, J. K. Lenneman, and C. L. Baldwin, "Advanced driver assistance systems (ADAS): Demographics, preferred sources of information, and accuracy of ADAS knowledge," *Transp. Res. F, Traffic Psychol. Behav.*, vol. 86, pp. 131–150, Apr. 2022.

[3] S. Lu and W. Shi, "Vehicle computing: Vision and challenges," *J. Inf. Intell.*, vol. 1, no. 1, pp. 23–35, Oct. 2022.

[4] N. Bhuyan and S. Moharir, "Multi-model federated learning," in *Proc. 14th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2022, pp. 779–783.

[5] S. Lu and W. Shi, "Vehicle as a mobile computing platform: Opportunities and challenges," *IEEE Netw.*, vol. 38, no. 6, pp. 493–500, Nov. 2024.

[6] R. Talat, M. S. Obaidat, M. Muzammal, A. H. Sodhro, Z. Luo, and S. Pirbhulal, "A decentralised approach to privacy preserving trajectory mining," *Future Gener. Comput. Syst.*, vol. 102, pp. 382–392, Jan. 2020.

[7] A. H. Sodhro et al., "Quality of service optimization in an IoT-driven intelligent transportation system," *IEEE Wireless Commun.*, vol. 26, no. 6, pp. 10–17, Dec. 2019.

[8] Y. Lin, X. Jin, J. Chen, A. H. Sodhro, and Z. Pan, "An analytic computation-driven algorithm for decentralized multicore systems," *Future Gener. Comput. Syst.*, vol. 96, pp. 101–110, Jul. 2019.

[9] A. Lakhan, M. A. Dootio, T. M. Groenli, A. H. Sodhro, and M. S. Khokhar, "Multi-layer latency aware workload assignment of E-transport IoT applications in mobile sensors cloudlet cloud networks," *Electronics*, vol. 10, no. 14, p. 1719, Jul. 2021.

[10] Y.-J. Cha, W. Choi, G. Suh, S. Mahmoudkhani, and O. Büyüköztürk, "Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types," *Comput-Aided Civ. Inf. Eng.*, vol. 33, no. 9, pp. 731–747, 2018.

[11] R. Ali, Q. Bin-Saeed, O. Büyüköztürk, S. Lee, and Y. Cha, "Monocular computer vision-based simultaneous pothole segmentation and 3D volume prediction using 3dpredictnet," *Available SSRN*, Jan. 2024.

[12] Y.-J. Cha, W. Choi, and O. Büyüköztürk, "Deep learning-based crack damage detection using convolutional neural networks," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 32, no. 5, pp. 361–378, May 2017.

[13] W. Choi and Y.-J. Cha, "SDDNet: Real-time crack segmentation," *IEEE Trans. Ind. Electron.*, vol. 67, no. 9, pp. 8016–8025, Sep. 2020.

[14] D. Kang and Y. Cha, "Autonomous UAVs for structural health monitoring using deep learning and an ultrasonic beacon system with geo-tagging," *Comput.-Aided Civil Infrastruct. Eng.*, vol. 33, no. 10, pp. 885–902, Oct. 2018.

[15] D. Kang, S. S. Benipal, D. L. Gopal, and Y.-J. Cha, "Hybrid pixel-level concrete crack segmentation and quantification across complex backgrounds using deep learning," *Autom. Construct.*, vol. 118, Oct. 2020, Art. no. 103291.

[16] X. Wang, A. Shrivastava, and A. Gupta, "A-Fast-RCNN: Hard positive generation via adversary for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 3039–3048.

[17] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Proc. 18th Int. Conf. Med. Image Comput. Comput.-Assist. Intervent.*, vol. 9351. Cham, Switzerland: Springer, 2015, pp. 234–241.

[18] Z. Wang, W. Ren, and Q. Qiu, "LaneNet: Real-time lane detection networks for autonomous driving," 2018, *arXiv:1807.01726*.

[19] S. Caldas et al., "LEAF: A benchmark for federated settings," 2018, *arXiv:1812.01097*.

[20] C. Li, C. Li, Y. Zhao, B. Zhang, and C. Li, "An efficient multi-model training algorithm for federated learning," in *Proc. IEEE Global Commun. Conf. Conf.*, Madrid, Spain, Dec. 2021, pp. 1–6.

[21] K. Kowsari, M. Heidarysafa, D. E. Brown, K. J. Meimandi, and L. E. Barnes, "RMDL: Random multimodel deep learning for classification," in *Proc. 2nd Int. Conf. Inf. Syst. Data Mining (ICISDM)*, Apr. 2018, pp. 19–28.

[22] Y. Jee Cho, J. Wang, and G. Joshi, "Client selection in federated learning: Convergence analysis and power-of-choice selection strategies," 2020, *arXiv:2010.01243*.

[23] Y. J. Cho, S. Gupta, G. Joshi, and O. Yağan, "Bandit-based communication-efficient client selection strategies for federated learning," in *Proc. IEEE 54th Asilomar Conf. Signals, Syst., Comput.*, Nov. 2020, pp. 1066–1069.

[24] W. Xia, T. Q. S. Quek, K. Guo, W. Wen, H. H. Yang, and H. Zhu, "Multi-armed bandit-based client scheduling for federated learning," *IEEE Trans. Wireless Commun.*, vol. 19, no. 11, pp. 7108–7123, Nov. 2020.

[25] M. Ribero and H. Vikalo, "Communication-efficient federated learning via optimal client sampling," 2020, *arXiv:2007.15197*.

[26] W. Chen, S. Horvath, and P. Richtarik, "Optimal client sampling for federated learning," 2020, *arXiv:2010.13723*.

[27] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency Comput., Pract. Exper.*, vol. 23, no. 2, pp. 187–198, Feb. 2011.

[28] K. Li, X. Tang, and K. Li, "Energy-efficient stochastic task scheduling on heterogeneous computing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 11, pp. 2867–2876, Nov. 2014.

[29] S. AlEbrahim and I. Ahmad, "Task scheduling for heterogeneous computing systems," *J. Supercomput.*, vol. 73, no. 6, pp. 2313–2338, Jun. 2017.

[30] D. Crankshaw et al., "Clipper: A low-latency online prediction serving system," in *Proc. 14th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2017, pp. 613–627.

[31] C. Zhang, M. Yu, W. Wang, and F. Yan, "MArk: Exploiting cloud services for cost-effective, SLO-aware machine learning inference serving," in *Proc. USENIX Annu. Tech. Conf.*, 2019, pp. 1049–1062.

[32] R. S. Kannan, L. Subramanian, A. Raju, J. Ahn, J. Mars, and L. Tang, "GrandSLAm: Guaranteeing SLAs for jobs in microservices execution frameworks," in *Proc. 14th EuroSys Conf.*, Mar. 2019, pp. 1–16.

[33] F. Romero, Q. Li, N. J. Yadwadkar, and C. Kozyrakis, "INFaaS: Managed & model-less inference serving," 2019, *arXiv:1905.13348*.

[34] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-IID data," 2018, *arXiv:1806.00582*.

[35] M. Mohri, G. Sivek, and A. T. Suresh, "Agnostic federated learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4615–4625.

[36] H. Wang, Z. Kaplan, D. Niu, and B. Li, "Optimizing federated learning on non-IID data with reinforcement learning," in *Proc. IEEE Conf. Comput. Commun.*, Jul. 2020, pp. 1698–1707.

[37] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, "Language models as zero-shot planners: Extracting actionable knowledge for embodied agents," in *Proc. Int. Conf. Mach. Learn.*, 2022, pp. 9118–9147.

[38] I. Singh et al., "ProgPrompt: Generating situated robot task plans using large language models," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2023, pp. 11523–11530.

[39] M. Ahn et al., "Do as I can, not as I say: Grounding language in robotic affordances," 2022, *arXiv:2204.01691*.

[40] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," 2022, *arXiv:2210.03629*.

[41] S. Li, E. Ngai, F. Ye, and T. Voigt, "Auto-weighted robust federated learning with corrupted data sources," *ACM Trans. Intell. Syst. Technol.*, vol. 13, no. 5, pp. 1–20, Oct. 2022.

[42] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. 20th Int. Conf. Artif. Intell. Statist.*, vol. 54, A. Singh and J. Zhu, Eds., Fort Lauderdale, FL, USA, 2017, pp. 1273–1282.

[43] K. Pillutla, S. M. Kakade, and Z. Harchaoui, "Robust aggregation for federated learning," 2019, *arXiv:1912.13445*.

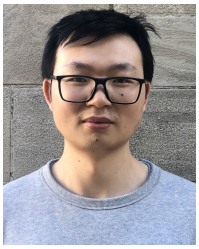
[44] F. Sattler, K.-R. Müller, T. Wiegand, and W. Samek, "On the Byzantine robustness of clustered federated learning," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, May 2020, pp. 8861–8865.

[45] P. Blanchard, E. M. E. Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, Dec. 2017, pp. 118–128.

[46] F. Yu et al., "BDD100K: A diverse driving dataset for heterogeneous multitask learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2633–2642.

[47] A. Jain, H. S. Koppula, B. Raghavan, S. Soh, and A. Saxena, "Car that knows before you do: Anticipating maneuvers via learning temporal driving models," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Dec. 2015, pp. 3182–3190.

Qiren Wang (Student Member, IEEE) received the B.S. degree in computer science from the University at Albany, SUNY, in 2020, and the M.S. degree in data science from Rutgers University in 2022. He is currently pursuing the Ph.D. degree in computer science with the University of Delaware, USA. His current research interests include autonomous driving, machine learning, federated learning, and edge computing.



Yongtao Yao is currently pursuing the Ph.D. degree with the University of Delaware. He is a member of the Connected and Autonomous Research (CAR) Laboratory. He has published several papers in prestigious journals and conferences and has a strong research foundation in applications within intelligent transportation and mobile computing. His research primarily focuses on computing systems with applications in autonomous driving, specifically in the areas of battery fault data diagnostics, task offloading, scheduling optimization, and the development and application of federated learning technologies.

Nejib Ammar received the Ph.D. degree in electrical and computer engineering from the University of California at Davis in 2005. He is a Senior Principal Scientist at Toyota North America. His team is currently working on connected vehicles, automotive cybersecurity, architecture, and algorithm design for communication systems.

Weisong Shi (Fellow, IEEE) is an Alumni Distinguished Professor and the Chair of the Department of Computer and Information Sciences, University of Delaware (UD), where he leads the Connected and Autonomous Research (CAR) Laboratory. He currently serves as the Honorary Director of the National Science Foundation (NSF) eCAT Industry-University Cooperative Research Center (IUCRC) for the term of 2023–2028, which focuses on advancing electric, connected, and autonomous mobility technologies. A globally recognized expert in edge computing, autonomous driving, and connected health, he has authored the pioneering paper "Edge Computing: Vision and Challenges," which has garnered over 8000 citations. Prior to joining UD, he held the position of a Professor at Wayne State University from 2002 to 2022. During his tenure, he held various administrative roles, including the Associate Dean for Research and Graduate Studies at the College of Engineering and the Interim Chair of the Computer Science Department. Additionally, he served as the National Science Foundation (NSF) Program Director from 2013 to 2015. He is currently the Editor-in-Chief of *IEEE Internet Computing Magazine* and *Smart Health* (Elsevier). He is also the Founding Steering Committee Chair of three conferences, including the ACM/IEEE Symposium on Edge Computing (SEC), the IEEE/ACM International Conference on Connected Health (CHASE), and the IEEE International Conference on Mobility (MOST). Notably, he is the General Chair of ACM MobiCom'24, the flagship conference on mobile computing and wireless networking. He is also a Distinguished Scientist of the Association for Computing Machinery (ACM), a member of the National Science Foundation (NSF) Computer and Information Science Engineering (CISE) Advisory Committee, and a Council Member of the Computing Community Consortium (CCC) under the Computer Research Association (CRA).