MCFL – Mixed Confidential Federated Learning over Unreliable Vehicular Networks

Guilherme A. Thomaz[1], Anna Glownia[2], Matteo Sammarco[3], and Miguel Elias M. Campista[1]
[1]*Federal University of Rio de Janeiro (UFRJ)*, Rio de Janeiro, Brazil
[2]*Stellantis*, Krakow, Poland. [3]*Stellantis*, Poissy, France
guiaraujo@gta.ufrj.br, anna.glownia@stellantis.com, matteo.sammarco@stallantis.com, miguel@gta.ufrj.br

***Abstract*—Federated Learning (FL) offers promising benefits for enhancing data privacy in the automotive industry. However, during early adoption, only high-end vehicles are expected to support on-board model training. Vehicular FL also faces challenges from intermittent connectivity and shutdowns, which can lead to wasted computations. This paper introduces Mixed Confidential Federated Learning (MCFL), a new approach for supporting the incremental deployment of FL in the automotive sector by introducing two key enhancements: (i) a privacy-preserving solution that allows constrained vehicles to offload training to Trusted Execution Environments (TEEs) at edge nodes, and (ii) a redesigned client/server state machine tailored to unreliable network conditions. Experimental results show that MCFL leads to a three-fold reduction in transfer times compared with the usual gRPC/HTTP2. Furthermore, MCFL increases the number of effective client contributions by 3x, delivering a level of communication efficiency and reliability not achieved by existing FL frameworks.**

***Index Terms*—Federated Learning, Vehicular Wireless Networks, Fault-Tolerant Communications, Trusted Execution Environments**

I. Introduction

Manufacturers can leverage data generated by connected vehicles to train Machine Learning (ML) models for complex tasks such as autonomous driving. To address privacy concerns, Federated Learning (FL) retains users' data locally on vehicles while exchanging only model updates with an aggregation server [1]. However, most mainstream vehicles either lack the computational resources to participate or must delegate training to the cloud. Mixed Federated Learning (MFL) combines training on capable client devices with training in the cloud [2], but considers that the cloud uses non-sensitive data. Confidential Learning (CL) enabled in-cloud training with sensitive data by using Trusted Execution Environments (TEEs), but it faces scalability and cost limitations when large volumes of data are processed centrally [3]. Hence, to the best of our knowledge, the joint use of MFL and CL to enhance privacy at scale in vehicular FL with heterogeneous hardware capabilities remains an open gap. Another overlooked limitation in both MFL and CL approaches is the assumption of uninterrupted client-server connectivity, which is unrealistic in vehicular environments [4]. Thus, we address the following question: *how to privately train ML models across large heterogeneous vehicular fleets with fault-tolerance?*

To answer this, we propose a framework called Mixed Confidential Federated Learning (MCFL). MCFL provides a meaningful step toward scalable, reliable, and privacy-preserving FL deployment for the automotive industry by enabling incremental adoption of resource-constrained and poorly connected vehicles. The main contributions of this paper are as follows. We propose, implement, and evaluate the MCFL framework based on real-world vehicular requirements: privacy preservation, support for heterogeneous and low-cost hardware, and robustness under unreliable connections. In addition, we allow resource-constrained vehicles to offload training to TEEs deployed at network edges. Only critical components are executed within the TEE, enabling training in affordable hardware. We introduce a novel Disconnection-Tolerant (DT) execution model that improves fault tolerance, leading to a threefold reduction in wasted computations. Finally, we implement DT messages using a non-persistent HTTP1.1 approach, resulting in 3x greater efficiency and 8% faster performance compared with typical gRPC/HTTP2.

The remainder of this work is organized as follows. Section II presents previous works. Section III describes MCFL architecture and implementation. Section IV presents our experiments and results, and Section V concludes this work.

II. Background and Related Work

This section discusses prior efforts to address privacy preservation, scalability, and network unreliability in FL for heterogeneous vehicular environments.

Mixed Federated Learning. In traditional FL, clients train a local model for a given number of epochs and send model updates to a centralized server for aggregation. The server generates a global model and redistributes it for the next round. However, low-end devices cannot participate, degrading overall model performance and generalization [2]. MFL mitigates this issue by incorporating public data on the server side. However, this design assumes that server-side data is non-sensitive, which does not hold for vehicular applications that handle private data.

Confidential Learning. Standard cryptographic methods are insufficient to secure data when it is uploaded to the cloud if privileged insiders exploit the entire operating system. Confidential computing isolates training inside a TEE. Intel SGX, for instance, creates isolated memory enclaves inaccessible to any external software [5]. By delegating model training to a TEE on edge servers, vehicles can achieve privacy guarantees

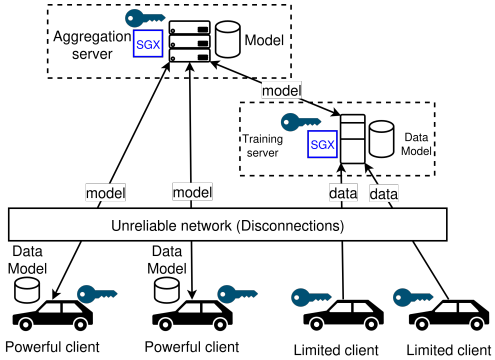


Fig. 1: Proposed Mixed Confidential Federated Learning (MCFL) framework.

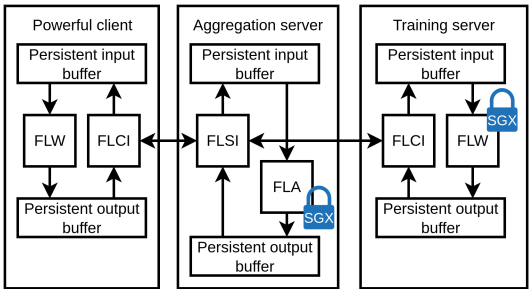


Fig. 2: MCFL modules.

similar to local training without overburdening onboard computing resources. Even if TEEs can protect client data in the cloud, full centralization introduces high infrastructure costs and energy consumption. A hybrid approach that distributes training between cloud TEEs and hardware-rich vehicles thus becomes appealing. Cai *et al.* demonstrate that using multiple enclaves can improve computational performance by reducing CPU and memory bottlenecks [3]. However, their solution assumes all enclaves operate within the same centralized cloud infrastructure, ignoring manufacturers' costs.

FL in Vehicular Networks. In vehicular FL, clients often rely on expensive mobile data plans, motivating research into communication-efficient FL at different protocol stack layers. FedComm [6] evaluates different protocol stacks (UDP, TCP, gRPC/HTTP2) in mobile networks. Amadeo *et al.* replace IP at the network core with Information Centric Network (ICN) for interconnecting the nodes at the edge with the aggregation server at the cloud [7].

State Machine Design in FL. FL frameworks generally assume a state machine in which clients maintain continuous connections to the server. We refer to this behavior as *Discard Disconnected Devices* (D3). In this approach, if a client disconnects mid-training, it must re-establish a connection and start over, discarding prior training progress. This model severely limits deployment in mobile networks, where transient connectivity is the norm. FedMingle is an FL framework that implements a new finite-state machine (FSM) that reduces the dependency on persistent connections during the training, allowing developers to use asynchronous communication protocols [8].

Unlike previous work, our proposal addresses both the need to offload high-volume private data from hardware-limited vehicles and the challenge of operating under intermittent connectivity. Thus, MCFL enables private, cost-affordable, and fault-tolerant FL deployment in vehicular environments.

III. MCFL – Architecture and Implementation

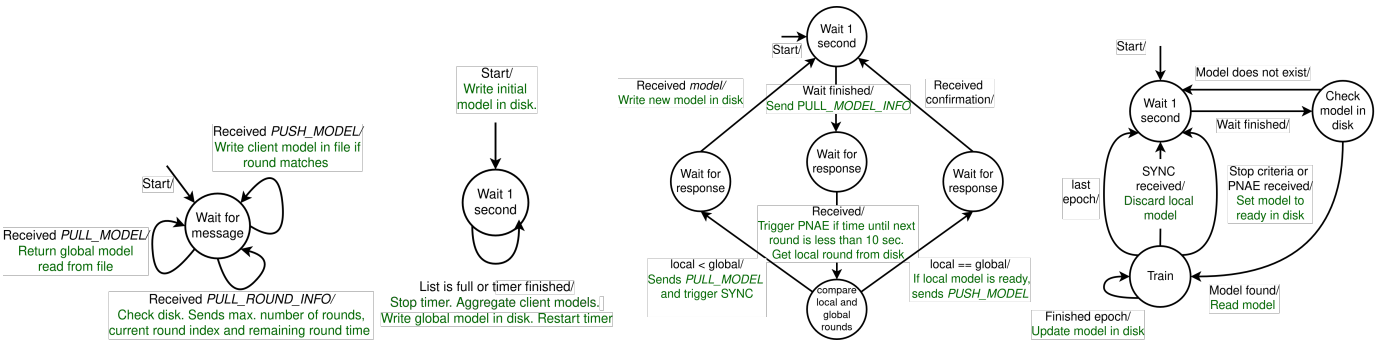
This section details the design choices of the proposed MCFL architecture and its implementation challenges. Figure 1 presents the MCFL architecture, composed of four main components: computationally-powerful clients, computationally-limited clients, an aggregation server, and one or more training servers. In each FL round, the aggregation server distributes the global model to both computationally-powerful clients and selected training servers. Powerful clients perform local training using their private data. In contrast, constrained clients upload their local data to training servers, which execute the training process within a TEE This offloading mechanism addresses heterogeneity in vehicular computing capabilities while maintaining strong privacy guarantees. Once training is complete, the model updates are securely returned to the aggregation server, which performs the aggregation step inside an enclaveTEEs are authenticated via remote attestation protocols, preventing attackers from running a malicious enclave to compromise the model. All persistent data and model checkpoints are sealed with keys accessible only within the enclave.

A. Software modules

The MCFL framework is organized into four software modules, illustrated in Figure 2. The Federated Learning Worker (FLW) executes local model training, while the Federated Learning Aggregator (FLA) implements the aggregation logic to combine model updates from clients or training servers. In addition, the Federated Learning Client Interface (FLCI) exchanges model parameters between the client and the aggregation server, while the Federated Learning Server Interface (FLSI) manages communication of model parameters between the aggregation server and training nodes. Computationally-powerful clients and training servers concurrently execute FLW and FLCI, while the aggregation server concurrently runs FLA and FLSI. The communication interfaces (FLCI and FLSI) are decoupled from the training and aggregation logic, allowing protocol adaptations or upgrades without impacting the core FL operations.

B. Disconnection-Tolerant Execution Model

Vehicular communication over 4G/5G is frequently disrupted by mobility-induced challenges such as signal shadowing, variable bandwidth, and high packet loss rates [6], [9]. In such environments, clients may disconnect unexpectedly and fail to reconnect. Standard FL approaches face a trade-off in this context, since the server must: (i) wait for reconnection, risking unacceptable latency; (ii) discard straggler updates,



(a) FLSI (b) FLA (c) FLCI (d) FLW

Fig. 3: Server (a and b) and client (c and d) state diagrams for Disconnection-Tolerant (DT) execution model.

wasting client-side computation and energy; or (iii) resort to asynchronous aggregation, often at the cost of degraded convergence. To address these issues, we replace the *Discard Disconnected Devices* (D3) execution model with a novel Disconnection-Tolerant (DT) execution model, whose finite state machines (FSMs) are seen in Figure 3. Each transition in the FSM is labeled A/B, where A triggers a state change and B represents the resulting response or action. The DT execution begins when the FLCI sends a `PULL_ROUND_INFO` message to the FLSI, including a unique client ID. If this is the client's first message, the server registers the ID and maintains a list of participating clients. The server responds with a `ROUND_INFO` message containing the current round index and the total number of planned rounds. For downloading the model, the FLCI issues a `PULL_MODEL` request to retrieve the updated model. This model replaces the client's local model and is written into the persistent input buffer (PIB). During training, the FLW writes the locally updated model into a persistent output buffer (POB), ensuring that progress is not lost if the vehicle is turned off. When training is complete, the FLCI sends the updated model to the FLSI (`PUSH_MODEL`), which writes it to its PIB. When all models are received, the FLA aggregates the updates and writes the new global model to the POB. Unlike D3, the DT model allows clients to disconnect during training. All communication is client-initiated, and the server never pushes updates to clients.

Upon publishing a new global model, FLA starts a round timer in a separate thread. If the timer expires before all models are received, the FLA limits the aggregation to the received models and proceeds to the next round. The timer is canceled early if all registered clients submit their updates. The `ROUND_INFO` message includes the remaining time for the current round (the FLA writes the value into disk and the FLSI reads it). Thus, by periodically polling the server with `PULL_ROUND_INFO`, the FLCI can estimate how much time remains. If the time falls below a predefined threshold, the FLCI signals the FLW to terminate the current epoch early. This mechanism, called Prior Notice with Adaptive Epochs (PNAE), ensures that slow clients contribute partial results rather than being ignored entirely, addressing the starvation issue in D3 models like that of Flower, and reducing wasted computation. If a client reconnects after the server has already advanced to a new round, the FLCI detects this by comparing the received round index in `ROUND_INFO` with its local round. In this case, it downloads the new global model and sends a `SYNC` signal to the FLW, instructing it to discard the stale model and restart training from the new round. HTTP `GET` and `POST` methods are used to implement `PULL` and `PUSH` messages, respectively. At the transport layer, each HTTP request requires the client to establish a new TCP connection, which inherently provides resilience.

C. Implementation Challenges

Intel SGX instructions are typically exposed via C libraries, which introduces considerable development overhead. To alleviate this overhead, we employ Gramine-SGX [10], a library OS that simplifies porting unmodified applications, including the Python interpreter and its ML libraries, into enclaves. Gramine-SGX achieves this by embedding a minimal OS abstraction layer inside the enclave. Since SGX enclaves cannot perform system calls directly, Gramine-SGX forwards operations such as file and network access to the host OS. These interactions are secured using encryption and integrity checks. Additionally, Gramine offers a component called Gramine-Direct, which runs the library OS outside enclaves, aiding development and benchmarking. Enclave execution introduces non-negligible performance overheads due to memory copying between untrusted and enclave memory regions [11]. Additionally, many Intel Skylake processors limit the enclave size to 128MB. When the enclave memory exceeds this threshold, pages are evicted to untrusted memory in encrypted form. Retrieving evicted pages further introduces latency due to the required decryption.

Adapting MCFL to run within the Gramine framework requires addressing the following challenges: (i) reducing the memory footprint of enclave code and data to optimize performance on resource-constrained devices with limited enclave size; (ii) minimizing the frequency of system calls made from within the enclave, which are costly due to untrusted and trusted context switching; and (iii) handling system calls related to OS signals and locks, which are unsupported in

TABLE I: Experiments setup

Parameter	Value
Desktop specs	i9-10900 CPUs, 32GB RAM, 20 threads
Confidence interval	95%
Layers	784 (ReLU), 200 (ReLU), 10 (linear)
Epochs and rounds	3 local epochs and 10 global rounds
Training	Cross Entropy loss + SGD $\eta = 0.05$
Aggregation	FedAvg
Dataset and task	MNIST for Digit classification
Client train samples	750 batches × 32 = 24k samples
Server test samples	188 batches × 32 ≈ 6k samples

TABLE II: Execution times (in ms) for each step in an MCFL FL round

Task	Baremetal	Gramine-Direct	Gramine-SGX
Client times			
Training (FLW)	13363.1 ± 0.1	15995.3 ± 0.1	130521.0 ± 0.1
Testing (FLW)	1054.3 ± 0.3	1218.1 ± 0.3	4441.36 ± 0.07
Load (FLW)	25 ± 1	17.5 ± 0.6	35.0 ± 1.0
Dump (FLW)	33.7 ± 0.9	35.5 ± 1	85.1 ± 0.7
Upload (FLCI)	1277.8 ± 0.8	1276.7 ± 0.7	1283.5 ± 0.3
Server times			
Aggregating (FLA)	0.55 ± 0.02	0.67 ± 0.03	2.0 ± 0.3
Testing (FLA)	1680 ± 10	2050 ± 5	2089 ± 15
Load (FLA)	18 ± 1	17.3 ± 0.8	29 ± 7
Dump (FLA)	32.0 ± 0.9	34 ± 1	40 ± 2
Download (FLSI)	1260 ± 2	1283.9 ± 0.3	1291.5 ± 0.3
Total round time	**18814**	**21873**	**139768**

Gramine. To mitigate these issues, we decouple the model operations (FLW and FLA) from the communication interfaces (FLCI and FLSI). The FLW and FLA modules are executed within Gramine enclaves using PyTorch, whereas communication modules (FLCI and FLSI) are executed outside the enclave. This decision reduces enclave memory consumption and yields the use of external libraries incompatible with Gramine, such as Flask and Requests for HTTP/1.1, without requiring source code modifications. Communication accounts for most I/O operations, which occur periodically. Thus, offloading this logic outside the enclave reduces the number of enclave system calls, improving performance. To address the lack of OS signal support in Gramine, we developed a lightweight signaling mechanism that uses files for inter-process notification. Signal emission is implemented by writing a flag to a file, while detection involves polling this file. This mechanism enables the FLCI to trigger `PNAE` and `SYNC` signals toward an FLW process running inside the enclave.

D. Security Analysis

The architecture is designed to ensure confidentiality and integrity of user data at all stages: on client devices, during transmission, and at the server. Also, it must guarantee that the training task being executed matches the one agreed upon by the clients, ensuring transparency regarding the intended use of users' data. The system assumes a strong adversarial model in which an attacker may gain root access to cloud servers, including control over the hypervisor and operating system. This threat model reflects scenarios in which a malicious insider or privileged user compromises the cloud infrastructure. Three critical attack vectors arise under this assumption:

- **unauthorized data access:** the attacker may attempt to extract sensitive information by inspecting the network, memory, or files from other processes during training;
- **model tampering and misuse:** the attacker may interfere with the FL training by sending forged models to clients (violating task agreement) or exploiting received updates from a specific client to infer private information;
- **directed white-box model inversion attacks:** the attacker may obtain the model parameters trained with the local dataset of an individual client for reconstructing the original training samples. Aggregation helps obfuscating model parameters, making model inversion less effective;

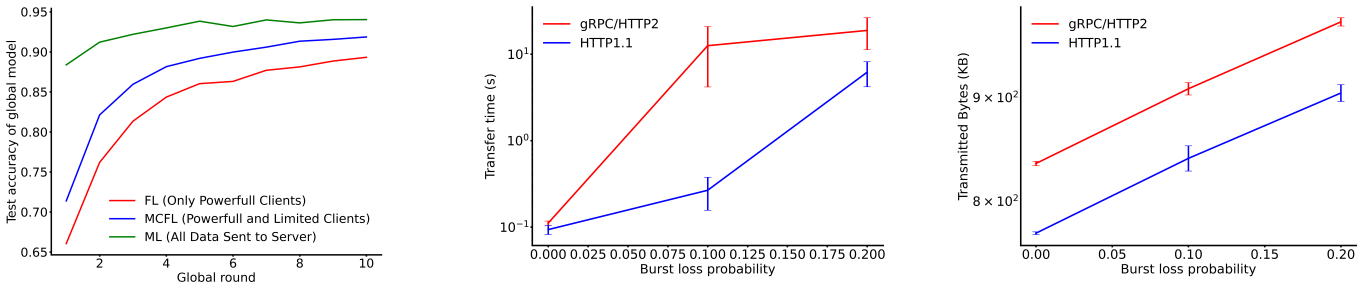
To provide security guarantees, the data must be decrypted only at the corresponding client or the training server's enclave, so that an attacker at the training server, regardless of its privilege level, cannot access the data being processed within the enclave's memory region. Regarding the network, data sent by limited clients is encrypted with a key exclusively shared with the training server's enclave. On the training server's disk, the data is encrypted with a sealing key only accessible to the enclave. The powerful clients and training servers receive the model from the aggregation server encrypted with a key shared only with the enclave, hindering attackers from impersonating the server and sending forged models. All the parameters sent to the aggregation server are also encrypted, which prevents white-box model inversion attacks directed to an specific client. Lastly, the establishment of a shared key with the enclave is performed via a remote attestation protocol, which includes verifying the genuineness of the code used to decrypt and process the confidential information. We consider physical and side-channel attacks out of scope [12].

IV. Experiments and Results

The details for the experiment are presented in Table I. We use two identical SGX-compatible physical desktops with enclave memory limited to 128 MB for measuring times. Virtual Machines were used to emulate additional FL clients.

A. Time Overhead

The first experiment evaluates the time taken by each step in a single MCFL round with two clients. Table II presents the mean and standard deviation (in milliseconds) over ten rounds. The baremetal column shows the baseline execution without Gramine, while the other two columns illustrate the overhead introduced by the Gramine library OS and the SGX enclave. In Table II, loading refers to the time taken to read and decrypt the model from disk, while dumping captures the time spent encrypting and writing the model back to disk. SGX introduces an overhead ranging from 18% to 157% for these operations, due to the need for system calls to perform I/O within enclaves. The upload and download times remain largely unaffected by Gramine and SGX, as they are measured at FLCI and FLSI, which run outside the enclave. The aggregation time is minimal due to the small number of clients, but this time is expected to increase linearly with the number of



(a) Model convergence. (b) Transfer time. (c) Bytes exchanged.

Fig. 4: Comparison between training approaches (a) and protocols (b and c).

participants. Overall, training and testing times dominate the total round duration. The Gramine library OS alone increases training and testing times by 20% and 16%, respectively. SGX further increases these durations by factors of 8x and 3.6×, respectively. This significant overhead is attributed to PyTorch's multi-threaded CPU backend, which parallelizes operations, increasing memory usage. As a result, the level of concurrency is reduced when enclave memory is constrained to 128 MB. Despite these limitations, MCFL confines only security-critical components, those that manipulate clear-text data, within SGX enclaves. The ability to train neural networks under such hardware constraints highlights the feasibility of MCFL as a cost-effective and privacy-preserving solution for FL in resource-constrained environments.

B. Influence of Limited Clients

Next, we compare the test accuracy of the global model in three scenarios: i) traditional FL, ii) MCFL, and iii) centralized ML (best case). We create 5 limited and 5 powerful clients with 600 training samples each. Figure 4a confirms that MCFL outperforms traditional FL, since limited clients send their data to an edge enclave, while still maintaining privacy. Although leading to lower accuracy than the fully centralized approach, the enclave in MCFL processes half the workload of centralized ML, enabling efficient training in low-cost machines.

C. Model Transference Overhead

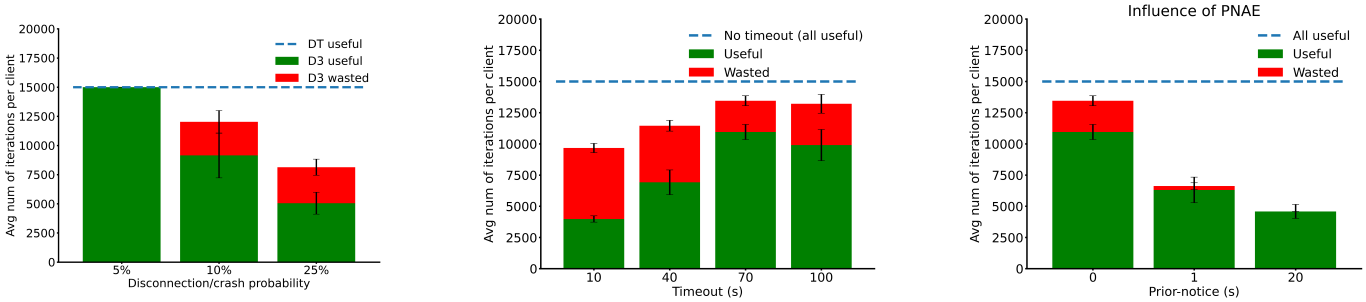
This experiment measures the average time required to transfer the model between client and server using gRPC/HTTP2 and HTTP/1.1. To emulate realistic network impairments, we configured NetEm to introduce constant packet losses based on the Gilbert-Elliot (GE) model, which consists of a two-state Markov chain, i.e., good and bad, where transitions from good to bad occur with probability p, and transitions from bad to good with probability $q = 1 - p$. Long-term packet loss rate tends toward p [13]. Figure 4b shows that HTTP/1.1 consistently outperforms gRPC/HTTP2 for all values of p. Analysis of PCAP traces reveals that gRPC employs HTTP2 `WINDOW_UPDATE` frames for application-layer flow control. While this mechanism is beneficial in scenarios involving concurrent HTTP2 streams over a single TCP connection, gRPC typically uses only one stream, making `WINDOW_UPDATE` counterproductive. Figure 4c shows the total number of bytes exchanged over the network under the same experimental conditions described in the previous section. By avoiding the flow control mechanisms of gRPC/HTTP2, HTTP/1.1 consistently reduces the network overhead, transmitting fewer bytes across all values of p.

D. Resilience to Disconnections

In the next experiment, a script launches four clients and a server as background processes. Every 10 seconds, the script simulates a client disconnection with probability p, and restarts the disconnected client with probability $1 - p$. To evaluate the system's robustness, we measure the average number of local backpropagation iterations each client performs over ten global rounds and ten local epochs, where each epoch consists of 150 iterations. We further distinguish between useful iterations, i.e., those incorporated into the server's aggregated model, and wasted iterations, i.e., those discarded due to a missed communication round caused by disconnection. As shown in Figure 5a, the DT execution model successfully incorporates all iterations into the training process for every value of p, reaching the theoretical maximum of 15k iterations, as indicated by the dashed horizontal line. In contrast, the D3 model fails to achieve this maximum when disconnections occur, and the proportion of wasted iterations increases with p. These results validate that DT's design offers superior resilience compared to D3 in environments with intermittent connectivity.

E. Timeout Influence

The previous experiment did not impose a timeout on the MCFL server, allowing all clients unlimited time to send their model updates. While this ensures that no contributions are lost, it may slow down global rounds in real-world scenarios, since a delayed client can stall the entire round. To evaluate this trade-off, we analyze the impact of the server timeout parameter (T) on the number of local iterations performed per client. To emulate client heterogeneity, before sending their updates, each of the 4 clients introduces a random



(a) Resilience to disconn. and crashes. (b) Timeout influence. (c) PNAE influence.

Fig. 5: Comparison of factors influencing the number of client iterations.

delay uniformly distributed between 0–20, 20–40, 40–60, and 60–80 seconds. Figure 5b presents the results. The horizontal dashed lines indicate the ideal scenario in which all clients successfully contribute their models. When the server timeout is set to $T = 10s$, slower clients frequently miss the deadline, resulting in numerous wasted iterations. As T increases, more clients are able to complete training and submit their models in time, leading to more useful iterations. However, this comes at the cost of longer global rounds.

F. PNAE/SYNC Influence

The previous experiments did not employ the PNAE mechanism described in Subsection III-B. As a result, local client models are discarded immediately upon server timeout, without any advance warning. In this next experiment, we fix the server timeout at $T = 70s$ and vary the PNAE timer to assess the effect of adapting the number of local epochs at clients on the number of wasted iterations. When the server issues a notification just 1 second before the timeout, the number of wasted iterations drops significantly. Increasing the PNAE timer further reduces wasted iterations, eventually reaching zero if all clients have sufficient time to stop training, persist the final model to the output buffer, and initiate transmission before the server times out. However, this improved efficiency comes at the cost of a reduced total number of local iterations.

V. Conclusion and Future Research

This paper proposes MCFL, an architecture that satisfies the combined requirements of privacy preservation, computational heterogeneity, and communication efficiency in vehicular networks. By allowing resource-constrained vehicles to offload training tasks to TEEs on the edge, MCFL significantly broadens participation. Moreover, MCFL minimizes TEE overhead, bringing training to affordable hardware. We also introduced the DT execution model. Experimental findings demonstrated a reduction in both communication latency and wasted local computations in unreliable networks. As future research, we plan to investigate the impact of client-side model inversion attacks and deploy MCFL in a real-world setting.

Acknowledgment

This study was financed, in part, by the São Paulo Research Foundation (FAPESP), Brazil, through project 2023/00811-0; Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) (Codes 001, 88887.954253/2024-00, and 88881.987122/2024-01) CNPq (310234/2025-5, 407304/2025-8, 408255/2023-4, and 405940/2022-0); FAPERJ (E-26/204.122/2024); FAPESP (2023/00673-7 and 2023/00811-0); Development and Research Foundation - Fundep - Rota 2030; and our partners Stellantis and Mobway.

References

[1] V. P. Chellapandi, L. Yuan, C. G. Brinton, S. H. Żak, and Z. Wang, "Federated learning for connected and automated vehicles: A survey of existing approaches and challenges," *arXiv*, vol. 9, no. 1, pp. 119–137, 2024.

[2] S. Augenstein, A. Hard, L. Ning, K. Singhal, S. Kale, K. Partridge, and R. Mathews, "Mixed Federated Learning: Joint Decentralized and Centralized Learning," Jun. 2022, arXiv:2205.13655 [cs].

[3] Z. Cai, B. Ren, R. Ma, H. Guan, M. Tian, and Y. Wang, "GUARDIAN: a hardware-assisted distributed framework to enhance deep learning security," *IEEE Transactions on Computational Social Systems*, vol. 10, no. 6, pp. 3012–3020, 2023, publisher: IEEE.

[4] X. Su, Y. Zhou, L. Cui, and J. Liu, "On model transmission strategies in federated learning with lossy communications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1173–1185, 2023.

[5] V. Costan and S. Devadas, "Intel SGX explained," *Cryptology ePrint Archive*, 2016.

[6] G. Cleland, D. Wu, R. Ullah, and B. Varghese, "FedComm: Understanding Communication Protocols for Edge-based Federated Learning," in *2022 IEEE/ACM 15th International Conference on Utility and Cloud Computing (UCC)*, Dec. 2022, pp. 71–81.

[7] M. Amadeo, C. Campolo, G. Ruggeri, and A. Molinaro, "Improving communication performance of Federated Learning: A networking perspective," *Computer Networks*, vol. 267, p. 111353, Jul. 2025. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128625003202

[8] N. Heydarishirayeh, "FedMingle: Communication in Federated Learning," Ph.D. dissertation, 2023.

[9] K. Brown and S. Singh, "M-tcp: Tcp for mobile cellular networks," *ACM SIGCOMM Computer Communication Review*, vol. 27, no. 5, pp. 19–43, 1997.

[10] C.-C. Tsai, D. E. Porter, and M. Vij, "{Graphene-SGX}: A practical library {OS} for unmodified applications on {SGX}," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 645–658.

[11] G. A. Thomaz, M. B. Guerra, M. Sammarco, M. Detyniecki, and M. E. M. Campista, "Tamper-proof access control for iot clouds using enclaves," *Ad Hoc Networks*, vol. 147, p. 103191, 2023.

[12] A. Nilsson, P. N. Bideh, and J. Brorsson, "A survey of published attacks on intel sgx," *arXiv preprint arXiv:2006.13598*, 2020.

[13] A. Bildea, O. Alphand, F. Rousseau, and A. Duda, "Link quality estimation with the gilbert-elliot model for wireless sensor networks," in *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE, 2015, pp. 2049–2054.