



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN  
AUTOMOTIVE ENGINEERING FOR INTELLIGENT MOBILITY

Ciclo 38

**Settore Concorsuale:** 09/F2 - TELECOMUNICAZIONI

**Settore Scientifico Disciplinare:** ING-INF/03 - TELECOMUNICAZIONI

DISTRIBUTED MACHINE LEARNING FOR 6G INTELLIGENT VEHICULAR  
COMMUNICATION NETWORKS

**Presentata da:** David Naseh

**Coordinatore Dottorato**

Davide Moro

**Supervisore**

Daniele Tarchi

**Co-supervisore**

Alessandro Vanelli Coralli

Esame finale anno 2025

# **DISTRIBUTED MACHINE LEARNING FOR 6G INTELLIGENT VEHICULAR COMMUNICATION NETWORKS**

**David Naseh**

**Department of Electrical, Electronic, and Information  
Engineering "Guglielmo Marconi" - DEI**

**University of Bologna**

**September 2025**



# **Distributed Machine Learning for 6G Intelligent Vehicular Communication Networks**

by

**David Naseh**

Department of Electrical, Electronic, and Information  
Engineering "Guglielmo Marconi" - DEI

Submitted

in partial fulfillment of the requirements of the degree of Doctor of Philosophy

to the

**University of Bologna**

**September 2025**





***This thesis is dedicated to***

***. . . my loving family . . .***

*for their unwavering love and support*

***. . . my esteemed supervisor, Daniele . . .***

*for his invaluable guidance and encouragement*

***. . . my cherished colleagues and friends . . .***

*for their constant companionship*

***throughout this journey.***



# Certificate

This is to certify that the thesis entitled “**Distributed Machine Learning for 6G Intelligent Vehicular Communication Networks**”, submitted by **Dr. David Naseh** to the University of Bologna, for the award of the degree of **Doctor of Philosophy** in Automotive for Intelligent Mobility, is a record of the original, bona fide research work carried out by him under our supervision and guidance. The thesis has reached the standards fulfilling the requirements of the regulations related to the award of the degree.

The results contained in this thesis have not been submitted in part or in full to any other University or Institute for the award of any degree or diploma to the best of our knowledge.

**Prof. Davide Moro**

(Coordinator of the PhD program of  
Automotive Engineering for Intelligent  
Mobility)  
Department of Industrial Engineering,  
University of Bologna.

**Prof. Daniele Tarchi**

(Supervisor)  
Department of Information Engineering,  
University of Florence.

**Prof. Alessandro Vanelli Coralli**

(Co-supervisor)  
Department of Electrical, Electronic,  
and Information Engineering  
"Guglielmo Marconi",  
University of Bologna.



# *Acknowledgements*

I would like to express my deepest gratitude to everyone who has supported me throughout my journey. Their guidance, encouragement, and generosity have been essential to the completion of this thesis and my growth as a scholar and individual.

First and foremost, I am profoundly grateful to my supervisor, Prof. Daniele Tarchi, whose encouragement, insightful feedback, and unwavering belief in me have been a constant source of motivation. His expertise and patience have not only guided me through the complexities of my research but also instilled a genuine passion for inquiry. I also extend my sincere thanks to my host supervisor at the University of Leeds, Prof. Arash Bozorgchenani, for his generous hospitality, access to cutting-edge facilities, and continuous intellectual support, which greatly enriched my work.

My heartfelt appreciation goes to Prof. Alessandro Vanelli Coralli and Prof. Giovanni Emanuele Corazza for welcoming me into the Digicomm research group and fostering a collaborative environment. I am also indebted to Prof. Davide Moro and Prof. Nicolò Cavina, coordinators of the PhD program, for their support in helping me pursue my research. I would like to further thank the reviewers of this thesis for their constructive comments, which significantly improved the quality of my work.

I have been fortunate to share this journey with many colleagues and friends. I owe special thanks to Swapnil for his unwavering support and insightful discussions, which were pivotal to my research. I also thank Mahdi, Rabih, and Bruno for their collaboration and encouragement within the Digicomm group. Outside the workplace, I am grateful to my friends Davod, Sadegh, and Mehrdad, whose support, laughter, and wisdom have enriched my life.

Finally, none of this would have been possible without the love and support of my family. To my siblings, especially my sister, whose kindness and helpfulness have brightened every step of this journey; and my brother Mehdi, whose guidance and generosity have shaped me in countless ways; and to my late parents, whose patience and belief in me laid the foundation for every achievement, I dedicate this work.

To all of you, I offer my sincerest thanks. Your generosity of spirit, wisdom, and friendship have been the greatest gifts of my doctoral journey.



# *Abstract*

The rapid advancements in 6G technologies are transforming Vehicular Networks (VNs) into smarter, more connected systems. As autonomous vehicles and Intelligent Transportation Systems (ITS) become central to modern society, ensuring efficient, scalable, and secure communication is paramount. This thesis addresses these challenges by integrating Distributed Learning (DL) techniques with Network Slicing (NS) and integrated Terrestrial and Non-Terrestrial Networks (T/NTNs) to satisfy the diverse and dynamic requirements of vehicular applications, such as Autonomous Driving (AD) and real-time traffic management, within the 6G ecosystem.

The primary **aim** is to develop an adaptive, scalable, and secure framework, DL-as-a-Service (DLaaS), that enables real-time data processing and ultra-low latency in 6G-enabled VNs. DLaaS unifies various DL techniques to optimize both communication and computation while preserving user privacy. It is designed to handle heterogeneous vehicular devices and the complexity of multilayer networks, ensuring seamless operation across integrated T/NTN layers.

**Methodologically**, the thesis introduces Federated Split Transfer Learning (FSTL) and its generalized version (GFSTL) to address challenges related to resource-constrained devices and heterogeneous network environments. These DL techniques are tailored to meet the needs of 6G VNs, offering scalable solutions to real-time learning, model synchronization, and privacy protection. The practical viability of these approaches is validated through real-world simulations and hardware implementations on Edge Computing (EC) platforms.

The key **findings** of this research demonstrate that the proposed DLaaS framework significantly enhances the accuracy, scalability, and latency of vehicular communication systems. Furthermore, the application of Deep Reinforcement Learning (DRL) for Dynamic Adaptive Streaming (DASH) in dynamic vehicular environments optimizes bitrate allocation, caching, and transcoding, improving Quality of Experience (QoE) for users in real-time. These results confirm the feasibility of integrating advanced Machine Learning (ML) techniques and T/NTNs into the design of 6G Internet of Vehicles (IoV) systems. In the end, proper conclusive remarks and several future directions are provided for the proposed solutions, offering valuable insights into the future of smart cities and intelligent mobility.





# Contents

Certificate

Acknowledgements

Abstract

Contents

List of Figures

List of Tables

List of Publications

Abbreviations

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>Introduction</b>                                   | <b>1</b> |
| 1.1      | Key Challenges . . . . .                              | 3        |
| 1.2      | Motivation . . . . .                                  | 4        |
| 1.3      | Contributions . . . . .                               | 5        |
| 1.4      | Organization . . . . .                                | 6        |
| <b>2</b> | <b>Network-Sliced Distributed Learning for 6G IoV</b> | <b>7</b> |
| 2.1      | Introduction . . . . .                                | 7        |
| 2.2      | IoV Distributed Intelligence . . . . .                | 12       |
| 2.2.1    | Edge/Cloud Computing . . . . .                        | 12       |
| 2.2.2    | Network Slicing . . . . .                             | 13       |
| 2.2.3    | Overview of Distributed Learning Methods . . . . .    | 14       |
| 2.2.4    | Intelligence at the Edge for IoV . . . . .            | 24       |
| 2.2.4.1  | Applications . . . . .                                | 24       |

|          |   |           |
|----------|---|-----------|
| 2.2.4.2  | Aspects and Advantages . . . . .  | 25        |
| 2.3      | Network Sliced Distributed Learning . . . . .   | 27        |
| 2.3.1    | End-to-End Functional Decomposition of DL . . . . .                                     | 27        |
| 2.3.2    | DL-as-a-Service for IoV Applications . . . . .  | 31        |
| 2.3.2.1  | Proposed Methods . . . . .  | 31        |
| 2.3.2.2  | DLaaS Advantages . . . . .  | 33        |
| 2.3.2.3  | DLaaS Challenges . . . . .  | 34        |
| 2.3.3    | DLaaS Architecture for IoV . . . . .  | 35        |
| 2.4      | Case Study . . . . .  | 38        |
| 2.4.1    | Scenario Description . . . . .  | 38        |
| 2.4.2    | Function Deployment . . . . .   | 39        |
| 2.4.3    | Network Resources and Simulation Parameters . . . . .                                   | 40        |
| 2.4.4    | Key Performance Indicators . . . . .  | 42        |
| 2.4.5    | Impact of Multiple Slices on User Satisfaction: . . . . .                               | 43        |
| 2.4.5.1  | Mobility . . . . .  | 44        |
| 2.4.5.2  | Processing capacity . . . . .   | 45        |
| 2.4.5.3  | Average Latency . . . . .   | 47        |
| 2.4.6    | Average Response Time . . . . .   | 48        |
| 2.5      | Conclusion . . . . .  | 49        |
| <b>3</b> | <b>Multi-Layer Distributed Learning for 6G ITS</b>                                      | <b>51</b> |
| 3.1      | Introduction . . . . .  | 52        |
| 3.1.1    | Gaps in the Literature and Motivations . . . . .  | 55        |
| 3.1.2    | Key Contributions . . . . .   | 58        |
| 3.1.3    | Organization of the Chapter . . . . .   | 61        |
| 3.2      | Distributed Learning Frameworks . . . . .   | 61        |
| 3.2.1    | Federated Learning . . . . .  | 62        |
| 3.2.2    | Split Learning . . . . .  | 63        |
| 3.2.3    | Federated Split Learning . . . . .  | 66        |
| 3.2.4    | Transfer Learning . . . . .   | 67        |
| 3.3      | Federated Split Transfer Learning . . . . .   | 69        |
| 3.3.1    | FSTL Architecture . . . . .   | 70        |
| 3.3.2    | FSTL Training Process . . . . .   | 72        |
| 3.3.3    | FSTL Advantages . . . . .   | 75        |
| 3.3.4    | FSTL in ITS Scenario . . . . .  | 75        |
| 3.4      | Generalized Federated Split Transfer Learning . . . . .                                 | 80        |
| 3.4.1    | GFSTL Architecture . . . . .  | 81        |
| 3.4.2    | GFSTL Training Process . . . . .  | 83        |
| 3.4.3    | GFSTL Advantages . . . . .  | 85        |
| 3.5      | Summary of Advantages and Disadvantages of Distributed Learning<br>Techniques . . . . . | 87        |
| 3.6      | Latency Analysis for Multilayer 6G ITS scenario . . . . .                               | 88        |

|          |   |            |
|----------|---|------------|
| 3.7      | Multilayer Distributed GFSTL Scenarios . . . . .  | 92         |
| 3.7.1    | Multilayer GFSTL in Vehicular Aerial-Ground Integrated Network . . . . .                            | 93         |
| 3.7.1.1  | System Architecture and Methodology . . . . .   | 93         |
| 3.7.1.2  | Advantages of Integrating GFSTL with Aerial Networks . . . . .                                      | 94         |
| 3.7.1.3  | GFSTL Workflow in Vehicular Scenarios . . . . .   | 95         |
| 3.7.2    | Multilayer GFSTL in NTN-based EO . . . . .  | 96         |
| 3.7.2.1  | Motivation and Proposed Framework . . . . .   | 97         |
| 3.7.2.2  | Framework Components and Advantages . . . . .   | 97         |
| 3.7.2.3  | GFSTL Training Process for EO over NTNs . . . . .   | 99         |
| 3.7.2.4  | Added Benefits and Challenges . . . . .   | 100        |
| 3.8      | Unified Hierarchical GFSTL for 6G ITS . . . . .   | 101        |
| 3.8.1    | Architecture and Components . . . . .   | 102        |
| 3.8.2    | Training Process and Iterative Algorithm . . . . .  | 103        |
| 3.8.3    | Added Benefits and Challenges . . . . .   | 105        |
| 3.9      | Simulations and Performance Evaluations . . . . .   | 107        |
| 3.9.1    | Simulation, Hardware, and Network Parameters . . . . .  | 108        |
| 3.9.2    | Model Convergence and Accuracy versus Rounds . . . . .  | 110        |
| 3.9.3    | User Diversity and Model Accuracy . . . . .   | 113        |
| 3.9.4    | Latency and Communication Efficiency . . . . .  | 115        |
| 3.9.5    | Summary of Key Results . . . . .  | 118        |
| 3.10     | Discussion . . . . .  | 120        |
| 3.11     | Conclusion . . . . .  | 121        |
| <b>4</b> | <b>Implementation and Performance Analysis of Distributed Learning Frameworks</b>                   | <b>123</b> |
| 4.1      | Introduction . . . . .  | 124        |
| 4.2      | Implementation and Evaluation of a Federated Learning Framework on Raspberry PI Platforms . . . . . | 125        |
| 4.2.1    | Technological Background . . . . .  | 127        |
| 4.2.2    | Contributions and Novelties . . . . .   | 129        |
| 4.2.3    | Limitations . . . . .   | 131        |
| 4.2.4    | Distributed Machine Learning . . . . .  | 131        |
| 4.2.5    | Implementation of the System . . . . .  | 136        |
| 4.2.5.1  | Server Configurations and Functionalities . . . . .   | 137        |
| 4.2.5.2  | Client Configurations and Functionalities . . . . .   | 140        |
| 4.2.6    | Simulations and Performance Evaluations . . . . .   | 143        |
| 4.2.6.1  | Effect of a Cooling Mechanism . . . . .   | 146        |
| 4.2.6.2  | Heterogeneous Client Compensation . . . . .   | 147        |
| 4.2.6.3  | Increasing the Number of Clients . . . . .  | 155        |
| 4.2.6.4  | Effect of Pretraining . . . . .   | 158        |

|          |   |            |
|----------|---|------------|
| 4.2.6.5  | Transfer Learning . . . . .   | 162        |
| 4.2.7    | Discussion . . . . .  | 163        |
| 4.2.8    | Conclusion . . . . .  | 165        |
| 4.3      | Real-World Implementation and Performance Analysis of Distributed Learning Frameworks for 6G IoT Applications . . . . . | 166        |
| 4.3.1    | Technological Background . . . . .  | 167        |
| 4.3.2    | Contributions and Novelties . . . . .   | 170        |
| 4.3.3    | Limitations . . . . .   | 171        |
| 4.3.4    | Distributed Learning . . . . .  | 172        |
| 4.3.4.1  | Federated Learning . . . . .  | 174        |
| 4.3.4.2  | Federated Transfer Learning . . . . .   | 178        |
| 4.3.5    | Implementation of the System . . . . .  | 181        |
| 4.3.5.1  | Server Configurations and Functionalities . . . . .   | 183        |
| 4.3.5.2  | Client Configurations and Functionalities . . . . .   | 184        |
| 4.3.6    | Experimental Results and Performance Evaluations . . . . .  | 188        |
| 4.3.6.1  | Experiment 1: DL with Five Heterogeneous Clients . . . . .  | 189        |
| 4.3.6.2  | Experiment 2: DL with Three Heterogeneous Clients . . . . .   | 205        |
| 4.3.7    | Conclusion . . . . .  | 217        |
| <b>5</b> | <b>Deep Reinforcement Learning for Dynamic Adaptive Streaming</b>   | <b>219</b> |
| 5.1      | Introduction . . . . .  | 219        |
| 5.1.1    | System Model and Problem Formulation . . . . .  | 222        |
| 5.1.2    | System Model . . . . .  | 222        |
| 5.1.3    | Problem Formulation . . . . .   | 224        |
| 5.2      | Proposed Solution . . . . .   | 227        |
| 5.2.1    | Preliminaries . . . . .   | 227        |
| 5.2.2    | DRL-based Solution for BrA-USA . . . . .  | 229        |
| 5.3      | Numerical Results . . . . .   | 232        |
| 5.3.1    | Simulation Setup . . . . .  | 233        |
| 5.3.2    | Simulation Results . . . . .  | 234        |
| 5.4      | Conclusion . . . . .  | 238        |
| <b>6</b> | <b>Conclusion</b>   | <b>239</b> |
|          | <b>Scope for Future Work</b>  | <b>242</b> |
|          | <b>Bibliography</b>   | <b>245</b> |

# List of Figures

|     |  |     |
|-----|--|-----|
| 2.1 | DLaaS over NS for IoV applications. . . . .  | 32  |
| 2.2 | Multi-layer Federated Learning as a slice for IoV scenarios. . . . .   | 33  |
| 2.3 | Slice and DL function deployment for regions with different IoV characteristics . . . . .  | 36  |
| 2.4 | User satisfaction in different KPIs vs the number of slices. . . . .   | 44  |
| 2.5 | User satisfaction with respect to mobility versus the number of slices for different number of VUs. . . . .                      | 45  |
| 2.6 | User satisfaction with respect to average processing capability versus the number of slices for different number of VUs. . . . . | 46  |
| 2.7 | User satisfaction with respect to average latency versus the number of slices for different number of VUs. . . . .               | 47  |
| 2.8 | Reconfigurability vs. the number of slices. . . . .  | 49  |
| 3.1 | Overall structure of FSTL . . . . .  | 71  |
| 3.2 | FSTL structure for ITS with RSU as the FSL Server . . . . .  | 77  |
| 3.3 | GFSTL architecture . . . . .   | 82  |
| 3.4 | GFSTL structure for vehicular AGIN. . . . .  | 94  |
| 3.5 | GFSTL structure for Earth Observation using Non-Terrestrial layers . . . . .   | 96  |
| 3.6 | Multilayer GFSTL Architecture for 6G ITS . . . . .   | 103 |
| 3.7 | Accuracy of different DL methods vs. number of training rounds. . . . .  | 112 |
| 3.8 | Accuracy of different DL methods vs. number of user batches per RSU. . . . .   | 114 |
| 3.9 | Latency of different DL methods vs. number of user batches per RSU. . . . .  | 116 |
| 4.1 | Considered FL framework with heterogeneous clients. . . . .  | 137 |
| 4.2 | Cooling mechanism for Raspberry Pi devices. . . . .  | 142 |
| 4.3 | Experimental setup used during the FL implementation. . . . .  | 143 |
| 4.4 | Accuracy of FL for 2 clients compared to the centralized benchmark vs. epochs. . . . .   | 145 |
| 4.5 | The effect of a cooling fan on the accuracy of training. . . . .   | 147 |
| 4.6 | Simulations with asymmetric data distribution, without and with random selection compared to the Centralized Benchmark. . . . .  | 149 |
| 4.7 | Accuracy with asymmetric distribution of data vs different numbers of local epochs. . . . .                                      | 150 |

|      |   |     |
|------|---|-----|
| 4.8  | Accuracy for asymmetric data distribution vs. different numbers of local epochs, in time domain. . . . .                      | 151 |
| 4.9  | Overfitting for different numbers of local epochs. . . . .  | 152 |
| 4.10 | Centralized Learning with and without random data selection. . . . .  | 153 |
| 4.11 | Federated Learning for 2 clients with and without random data selection. . . . .  | 154 |
| 4.12 | FL with different amounts of randomly chosen samples for two clients. . . . .   | 155 |
| 4.13 | FL with different amounts of randomly distributed samples among different number of clients. . . . .                          | 156 |
| 4.14 | Accuracy vs. the number of rounds for different numbers of clients. . . . .   | 157 |
| 4.15 | Accuracy with random samples and different numbers of clients. . . . .  | 158 |
| 4.16 | Accuracy with 2 clients and different pretraining levels. . . . .   | 159 |
| 4.17 | Simulations with 5 clients and different pretraining levels. . . . .  | 160 |
| 4.18 | Two simulations with 10 clients with or without pretraining vs. simulation with 2 clients without pretraining. . . . .        | 161 |
| 4.19 | TL with 2 clients. . . . .  | 162 |
| 4.20 | TL with 3 clients. . . . .  | 163 |
| 4.21 | Considered distributed framework with heterogeneous clients. . . . .  | 182 |
| 4.22 | Experimental setup used during the first DL implementations. . . . .  | 188 |
| 4.23 | Accuracy of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. rounds. . . . . | 190 |
| 4.24 | Accuracy of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. time. . . . .   | 191 |
| 4.25 | Loss of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. rounds. . . . .     | 193 |
| 4.26 | Loss of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. time. . . . .       | 194 |
| 4.27 | Average load on the Raspberry Pis and Odroid when running FTL and FL with 5 clients. . . . .                                  | 195 |
| 4.28 | Raspberry Pis and Odroid temperature when running FTL compared to FL with 5 clients. . . . .                                  | 197 |
| 4.29 | Memory usage of FTL compared to ordinary FL for 5 clients. . . . .  | 199 |
| 4.30 | Power consumption of Raspberry Pis and Odroid when running FTL and FL with 5 clients. . . . .                                 | 201 |
| 4.31 | Energy consumption of Raspberry Pi 1 when running FTL and FL. . . . .   | 203 |
| 4.32 | Energy consumption of Raspberry Pi 2 when running FTL and FL. . . . .   | 204 |
| 4.33 | Energy consumption of Odroid when running FTL and FL. . . . .   | 205 |
| 4.34 | Accuracy of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. rounds. . . . .  | 207 |
| 4.35 | Accuracy of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. time. . . . .    | 208 |

|      |  |     |
|------|--|-----|
| 4.36 | Loss of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. rounds. . . . . | 209 |
| 4.37 | Loss of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. time. . . . .   | 210 |
| 4.38 | Average load on the Raspberry Pi and Odroid when running FTL and FL with 3 clients. . . . .                              | 212 |
| 4.39 | Raspberry Pi and Odroid temperature when running FTL compared to FL with 3 clients. . . . .                              | 213 |
| 4.40 | Memory usage of FTL compared to ordinary FL for 3 clients. . . . .   | 214 |
| 4.41 | Power consumption of Raspberry Pi and Odroid when running FTL and FL with 3 clients. . . . .                             | 215 |
| 4.42 | Energy consumption of Raspberry Pi when running FTL and FL. . . . .  | 216 |
| 4.43 | Energy consumption of Odroid when running FTL and FL. . . . .  | 217 |
| 5.1  | Our considered architecture . . . . .  | 231 |
| 5.2  | Bitrate allocation, transcoding and joint rewards vs number of episodes  | 235 |
| 5.3  | (a) Bitrate error and (b) Root Mean Square of Bitrate error vs episodes  | 236 |
| 5.5  | Streaming source vs number of episodes . . . . .   | 237 |
| 5.4  | Transcoding percentage over the edge vs number of episodes . . . . .   | 237 |





# List of Tables

|     |   |     |
|-----|---|-----|
| 2.1 | Advantages, challenges, conditions and KPIs for different learning paradigms. . . . .   | 15  |
| 2.2 | Network resource allocation across layers. . . . .                                      | 41  |
| 3.1 | Comparison of the most related works and gaps/differences relative to our work. . . . . | 56  |
| 3.2 | Comparison of DL Techniques: Advantages and Disadvantages. . . .                        | 89  |
| 3.3 | Latency analysis of five DL methods per round. . . . .                                  | 92  |
| 3.4 | Simulation, hardware, and network parameters. . . . .                                   | 111 |
| 3.5 | Summary of results. . . . .   | 119 |
| 4.1 | Hardware specifications . . . . .   | 187 |



# List of Publications

## Journals

1. **David Naseh**, Arash Bozorgchenani, Swapnil Sadashiv Shinde, and Daniele Tarchi, "Unlocking Distributed Intelligence: A Comprehensive Survey on Federated Split Learning's Evolution, Challenges, and Future Frontiers." Submitted to Computer Networks.
2. Arash Bozorgchenani, **David Naseh**, Daniele Tarchi, Sergio A. Salinas Monroy, Farshad Mashhadi, and Ni, Qiang, "Reinforcing Edge-DASH: Deep Reinforcement Learning for Multi-Objective Streaming Optimization." Submitted to the IEEE Transactions on Mobile Computing (under revision).
3. **David Naseh**, Arash Bozorgchenani, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Unified Distributed Machine Learning for 6G Intelligent Transportation Systems: A Hierarchical Approach for Terrestrial and Non-Terrestrial Networks." Network 5, no. 3 (2025): 41.
4. Marius Corici, Marius Caus, Xavier Artiga, Alessandro Guidotti, Benjamin Barth, Tomaso Decola, Justin Tallon, Hemant Zope, Daniele Tarchi, Fanny Parzysz, **David Naseh**, and Swapnil Sadashiv Shinde, "Transforming 5G Mega-Constellation Communications: A Self-Organized Network Architecture Perspective," IEEE ACCESS, 2025, 13, Article number: 10844267, pp. 14770–14788.
5. **David Naseh**, Mahdi Abdollahpour, and Daniele Tarchi. "Real-World Implementation and Performance Analysis of Distributed Learning Frameworks for 6G IoT Applications." Information 15, no. 4 (2024): 190.

6. **David Naseh**, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Network Sliced Distributed Learning-as-a-Service for Internet of Vehicles Applications in 6G Non-Terrestrial Network Scenarios." *Journal of Sensor and Actuator Networks* 13, no. 1 (2024): 14.
7. Lorenzo Ridolfi, **David Naseh**, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Implementation and Evaluation of a Federated Learning Framework on Raspberry PI Platforms for IoT 6G Applications." *Future Internet* 15, no. 11 (2023): 358.
8. Niccolò Girelli Consolaro, Swapnil Sadashiv Shinde, **David Naseh**, and Daniele Tarchi. "Analysis and performance evaluation of transfer learning algorithms for 6G wireless networks." *Electronics* 12, no. 15 (2023): 3327.

## Conferences

1. **David Naseh**, Arash Bozorgchenani, and Daniele Tarchi. "Deep Reinforcement Learning for Edge-DASH-Based Dynamic Video Streaming." In 2025 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1-6. IEEE, 2025.
2. Swapnil Sadashiv Shinde, **David Naseh**, Tomaso De Cola, Daniele Tarchi, "A Distributed Task Allocation Methodology for Edge Computing in a LEO Satellite IoT Context." In 2025 the 12th Advanced Satellite Multimedia Systems Conference and the 18th Signal Processing for Space Communications Workshop (ASMS/SPSC), pp. 1-7. IEEE, 2025.
3. Swapnil Sadashiv Shinde, **David Naseh**, and Daniele Tarchi. "ML-Based Intelligent O-RAN Control in 6G Integrated Terrestrial and Non-Terrestrial Networks." In 2024 IEEE Globecom Workshops (GC Wkshps), pp. 1-6. IEEE, 2025.
4. **David Naseh**, Swapnil Sadashiv Shinde, Daniele Tarchi, and Tomaso De Cola. "Distributed Intelligent Framework for Remote Area Observation on Multilayer Non-Terrestrial Networks." In 2024 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), pp. 1-6. IEEE, 2024.

5. **David Naseh**, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Multi-layer distributed learning for intelligent transportation systems in 6G aerial-ground integrated networks." In 2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit), pp. 711-716. IEEE, 2024.
6. **David Naseh**, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Distributed learning framework for earth observation on multilayer non-terrestrial networks." In 2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN), pp. 1-2. IEEE, 2024.
7. Swapnil Sadashiv Shinde, **David Naseh**, and Daniele Tarchi. "In-Space Computation Offloading for Multi-layer LEO Constellations." In European Wireless 2023; 28th European Wireless Conference, pp. 82-89. VDE, 2023.
8. **David Naseh**, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Enabling Intelligent Vehicular Networks Through Distributed Learning in the Non-Terrestrial Networks 6G Vision." In European Wireless 2023; 28th European Wireless Conference, pp. 136-141. VDE, 2023.

## **Others**

1. **David Naseh**, "Driving into the Future: Safer Roads Through Smarter Communication," Three-Minute-Thesis Competition (**Finalist**), University of Bologna, Bologna, Italy, March 2025.

Online Access through the University of Bologna YouTube Channel: <https://youtu.be/IJaLuUFvSN0?si=2zmMvKZAUl-BE3IU>

2. **David Naseh**, Swapnil Sadashiv Shinde, Daniele Tarchi, "Multi-Layer Network Sliced Distributed Learning for Satellite IoT," IEEE Workshop on Innovative Ideas for Satellite 6G, Siena, Italy, April 2023.

Online Access through IEEE Future Networks: <https://ieeetv.ieee.org/channels/ieee-future-networks/multi-layer->



# Abbreviations

|              |   |
|--------------|---|
| <b>5G</b>    | <b>5th Generation</b>                             |
| <b>6G</b>    | <b>6th Generation</b>                             |
| <b>3GPP</b>  | <b>3rd Generation Partnership Project</b>         |
| <b>ABR</b>   | <b>Adaptive Bit Rate</b>                          |
| <b>AD</b>    | <b>Autonomous Driving</b>                         |
| <b>ADAS</b>  | <b>Advanced Driver Assistance Systems</b>         |
| <b>AGIN</b>  | <b>Aerial-Ground Integrated Network</b>           |
| <b>AI</b>    | <b>Artificial Intelligence</b>                    |
| <b>B5G</b>   | <b>Beyond 5th Generation</b>                      |
| <b>BA</b>    | <b>Bandwidth Allocation</b>                       |
| <b>BrA</b>   | <b>Bitrate Allocation</b>                         |
| <b>BS</b>    | <b>Base Station</b>                               |
| <b>BSP</b>   | <b>Bitrate Selection Point</b>                    |
| <b>CFL</b>   | <b>Collaborative Federated Learning</b>           |
| <b>CL</b>    | <b>Centralized Learning</b>                       |
| <b>CO</b>    | <b>Caching Optimization</b>                       |
| <b>CPU</b>   | <b>Central Processing Unit</b>                    |
| <b>CVIS</b>  | <b>Cooperative Vehicle Infrastructure Systems</b> |
| <b>DAF</b>   | <b>Data Acquisition Function</b>                  |
| <b>DASH</b>  | <b>Dynamic Adaptive Streaming over HTTP</b>       |
| <b>DCF</b>   | <b>Data Collection Function</b>                   |
| <b>DDPG</b>  | <b>Deep Deterministic Policy Gradient</b>         |
| <b>DIL</b>   | <b>Distributed Incremental Learning</b>           |
| <b>DL</b>    | <b>Distributed Learning</b>                       |
| <b>DLaaS</b> | <b>Distributed Learning-as-a-Service</b>          |
| <b>DLF</b>   | <b>Distributed Learning Function</b>              |



|               |   |
|---------------|---|
| <b>DMIF</b>   | <b>D</b> istributed <b>M</b> odel <b>I</b> nference <b>F</b> unction                    |
| <b>DNN</b>    | <b>D</b> eep <b>N</b> eural <b>N</b> etwork   |
| <b>DP</b>     | <b>D</b> ecision <b>P</b> oint  |
| <b>DPrF</b>   | <b>D</b> ata <b>P</b> re-processing <b>F</b> unction                                    |
| <b>DPsF</b>   | <b>D</b> ata <b>P</b> ost-processing <b>F</b> unction                                   |
| <b>DQN</b>    | <b>D</b> eep <b>Q</b> - <b>N</b> etwork   |
| <b>DRL</b>    | <b>D</b> eep <b>R</b> einforcement <b>L</b> earning                                     |
| <b>EC</b>     | <b>E</b> dge <b>C</b> omputing  |
| <b>ECs</b>    | <b>E</b> dge <b>C</b> lients  |
| <b>EO</b>     | <b>E</b> arth <b>O</b> bservation   |
| <b>FBA</b>    | <b>F</b> air <b>B</b> andwidth <b>A</b> llocation                                       |
| <b>FedAvg</b> | <b>F</b> ederated <b>A</b> veraging   |
| <b>FD</b>     | <b>F</b> ederated <b>D</b> istillation  |
| <b>FL</b>     | <b>F</b> ederated <b>L</b> earning  |
| <b>FLOPS</b>  | <b>F</b> loating <b>P</b> oint <b>O</b> perations <b>P</b> er <b>S</b> econd            |
| <b>FSL</b>    | <b>F</b> ederated <b>S</b> plit <b>L</b> earning  |
| <b>FSTL</b>   | <b>F</b> ederated <b>S</b> plit <b>T</b> ransfer <b>L</b> earning                       |
| <b>FTL</b>    | <b>F</b> ederated <b>T</b> ransfer <b>L</b> earning                                     |
| <b>GADMM</b>  | <b>G</b> roup <b>A</b> lternate <b>D</b> irection <b>M</b> ethod of <b>M</b> ultipliers |
| <b>Gb</b>     | <b>G</b> iga <b>b</b> it  |
| <b>GB</b>     | <b>G</b> iga <b>B</b> yte   |
| <b>GD</b>     | <b>G</b> radient <b>D</b> escent  |
| <b>GFSTL</b>  | <b>G</b> eneralized <b>F</b> ederated <b>S</b> plit <b>T</b> ransfer <b>L</b> earning   |
| <b>GMUF</b>   | <b>G</b> lobal <b>M</b> odel <b>U</b> pdate <b>F</b> unction                            |
| <b>GPU</b>    | <b>G</b> raphical <b>P</b> rocessing <b>U</b> nit                                       |
| <b>HAP</b>    | <b>H</b> igh <b>A</b> ltitude <b>P</b> latforms   |
| <b>IoV</b>    | <b>I</b> nternet <b>o</b> f <b>V</b> ehicles  |
| <b>IoT</b>    | <b>I</b> nternet <b>o</b> f <b>T</b> hings  |
| <b>ITS</b>    | <b>I</b> ntelligent <b>T</b> ransportation <b>S</b> ystem                               |
| <b>KD</b>     | <b>K</b> nowledge <b>D</b> istillation  |
| <b>KPI</b>    | <b>K</b> ey <b>P</b> erformance <b>I</b> ndicator                                       |
| <b>KNN</b>    | <b>k</b> - <b>N</b> earest <b>N</b> eighbor   |
| <b>KT</b>     | <b>K</b> nowledge <b>T</b> ransfer  |
| <b>LAP</b>    | <b>L</b> ow <b>A</b> ltitude <b>P</b> latforms  |
| <b>LEO</b>    | <b>L</b> ow <b>E</b> arth <b>O</b> rbital   |

|               |   |
|---------------|---|
| <b>LRU</b>    | <b>Least Recently Used</b>                    |
| <b>LTE</b>    | <b>Long-Term Evolution</b>                    |
| <b>MARL</b>   | <b>Multi-Agent Reinforcement Learning</b>     |
| <b>MBS</b>    | <b>Macro Base Station</b>                     |
| <b>MINLP</b>  | <b>Mixed Integer NonLinear Programming</b>    |
| <b>MEC</b>    | <b>Multi-access Edge Computing</b>            |
| <b>MES</b>    | <b>Macro Edge Server</b>                      |
| <b>ML</b>     | <b>Machine Learning</b>                       |
| <b>NF</b>     | <b>Network Function</b>                       |
| <b>NFV</b>    | <b>Network Function Virtualization</b>        |
| <b>NN</b>     | <b>Neural Network</b>                         |
| <b>NS</b>     | <b>Network Slicing</b>                        |
| <b>NTN</b>    | <b>Non-Terrestrial Networks</b>               |
| <b>OS</b>     | <b>Operating System</b>                       |
| <b>PDF</b>    | <b>Probability Density Function</b>           |
| <b>QoE</b>    | <b>Quality of Experience</b>                  |
| <b>QoS</b>    | <b>Quality of Service</b>                     |
| <b>RAM</b>    | <b>Random Access Memory</b>                   |
| <b>RIS</b>    | <b>Reconfigurable Intelligent Surfaces</b>    |
| <b>RL</b>     | <b>Reinforcement Learning</b>                 |
| <b>RMSBRE</b> | <b>Root Mean Square of Bit-Rate Error</b>     |
| <b>RSU</b>    | <b>Road-Side Unit</b>                         |
| <b>SBS</b>    | <b>Small Base Station</b>                     |
| <b>SDN</b>    | <b>Software-Defined Networking</b>            |
| <b>SES</b>    | <b>Small Edge Server</b>                      |
| <b>SGD</b>    | <b>Stochastic Gradient Descent</b>            |
| <b>SINR</b>   | <b>Signal-to-Inference-plus-Noise Ratio</b>   |
| <b>SL</b>     | <b>Split Learning</b>                         |
| <b>SVC</b>    | <b>Scalable Video Coding</b>                  |
| <b>TD</b>     | <b>Temporal Difference</b>                    |
| <b>TL</b>     | <b>Transfer Learning</b>                      |
| <b>TN</b>     | <b>Terrestrial Network</b>                    |
| <b>T/NTNs</b> | <b>Terrestrial / Non-Terrestrial Networks</b> |
| <b>TPU</b>    | <b>Tensor Processing Unit</b>                 |
| <b>UAV</b>    | <b>Unmanned Aerial Vehicle</b>                |

|             |  |
|-------------|--|
| <b>USA</b>  | <b>U</b> ser to <b>S</b> erver <b>A</b> llocation    |
| <b>V2I</b>  | <b>V</b> ehicle- <b>to</b> - <b>I</b> nfrastructure  |
| <b>V2P</b>  | <b>V</b> ehicle- <b>to</b> - <b>P</b> erson          |
| <b>V2R</b>  | <b>V</b> ehicle- <b>to</b> - <b>R</b> oad Side Units |
| <b>V2S</b>  | <b>V</b> ehicle- <b>to</b> - <b>S</b> ensors         |
| <b>V2V</b>  | <b>V</b> ehicle- <b>to</b> - <b>V</b> ehicle         |
| <b>V2X</b>  | <b>V</b> ehicle- <b>to</b> - <b>E</b> verything      |
| <b>VEC</b>  | <b>V</b> ehicular <b>E</b> dge <b>C</b> omputing     |
| <b>VMs</b>  | <b>V</b> irtual <b>M</b> achines                     |
| <b>VN</b>   | <b>V</b> ehicular <b>N</b> etwork                    |
| <b>VNC</b>  | <b>V</b> irtual <b>N</b> etwork <b>C</b> omputing    |
| <b>VU</b>   | <b>V</b> ehicular <b>U</b> ser                       |
| <b>YOLO</b> | <b>Y</b> ou <b>O</b> nly <b>L</b> ook <b>O</b> nce   |

# Chapter 1

## Introduction

The rapid advancement of fifth-generation (5G) wireless systems has catalyzed a paradigm shift in vehicular communications, ushering in the Internet of Vehicles (IoV) era [1]. As the industry anticipates the emergence of sixth-generation (6G) networks, expectations have grown for seamless global coverage, Ultra-Reliable Low-Latency Communications (URLLC), and pervasive intelligence embedded across the network edge [2]. Autonomous Driving (AD), cooperative sensing, real-time traffic control, and immersive infotainment represent only a fraction of the envisioned 6G-IoV applications, each imposing stringent requirements on end-to-end latency, reliability, and data privacy. Traditional centralized Machine Learning (ML) approaches, which aggregate raw data in the cloud for model training, face insurmountable challenges: bandwidth limitations hinder the uploading of massive sensor streams, privacy regulations restrict raw data sharing, and the heterogeneity of vehicular devices creates Non-Independent and Identically Distributed (Non-IID) data that degrade model performance [3]. These limitations motivate a decentralized learning paradigm that leverages the computation and data residing at the network edge.

Simultaneously, foundational developments in Distributed Learning (DL), most notably Federated Learning (FL), Split Learning (SL), and Transfer Learning (TL), promise to address these challenges by orchestrating training across geographically dispersed devices without exposing raw data [4]. FL enables collaborative model updates, SL partitions deep networks between clients and servers, and TL allows pretrained representations to adapt rapidly to new domains. Coupling these learning techniques with Network Slicing (NS), a fundamental feature of 5G and beyond, enables tailored isolation of compute, storage, and communication resources for distinct IoV services [5]. Furthermore, the integration of Non-Terrestrial Networks (NTNs), such as High-Altitude Platforms (HAPs) and Low Earth Orbit (LEO) satellites, can extend coverage and serve as additional aggregation points for DL in rural or disaster-affected areas [6].

This dissertation articulates a cohesive framework that synthesizes NS, joint Terrestrial and Non-Terrestrial Networks (T/NTNs), and advanced DL methods to meet the data privacy, latency, and reliability demands of 6G-IoV applications. We propose DL-as-a-Service (DLaaS) to virtualize learning workflows within network slices; Federated Split Transfer Learning (FSTL) and its generalization (GFSTL) to balance computation and communication across Vehicular Users (VUs), Road-Side Units (RSUs), and aerial aggregators; and Deep Reinforcement Learning (DRL)-based controllers for Dynamic Adaptive Streaming over HTTP (DASH) in Edge-Cloud three-tier environments. Through both simulation and hardware prototypes, our work demonstrates the feasibility and benefits of this hierarchical DL ecosystem.

## 1.1 Key Challenges

Deploying DL in Vehicular Networks (VNs) entails a confluence of technical challenges. First, data heterogeneity arises from Non-IID samples across vehicles. Each node collects distinct sensor modalities—camera, LiDAR, RADAR—under varying environmental conditions, which can bias local updates and hamper global model convergence [7]. Mitigating this requires aggregation strategies resilient to skewed distributions and client selection policies that ensure representative participation.

Second, vehicular devices are resource-constrained. Onboard computation units, while increasingly powerful, remain limited in processing capacity, memory footprint, and energy budget compared to cloud infrastructure. Consequently, Deep Neural Networks (DNNs) must be partitioned or compressed to fit device capabilities without sacrificing inference accuracy [8]. Techniques such as SL and model pruning must be adapted to dynamic vehicular contexts where system resources fluctuate with workload and mobility.

Third, connectivity in IoV is inherently intermittent. High vehicular speeds, urban canyons, and handovers between Terrestrial and Non-Terrestrial (T/NT) links generate unpredictable bandwidth and latency profiles [9]. Real-time applications—collision avoidance, platooning—demand end-to-end delays on the order of milliseconds. Orchestrating DL under such constraints necessitates flexible scheduling, adaptive split points, advanced slicing methods, and fallback mechanisms when connectivity degrades [10].

Fourth, privacy and security are paramount. Vehicular data often contains sensitive information—license plates, face imagery, geolocation traces—that risks exposure in centralized training. DL mitigates raw data transmission, but model updates can still leak private attributes through inference attacks. Robust privacy-preserving

mechanisms, including differential privacy and secure aggregation, must be integrated into the training pipeline without incurring prohibitive communication or computation overhead [11].

Finally, managing heterogeneous infrastructure—ranging from RSUs and Multi-access Edge Computing (MEC) nodes to satellites—requires unified orchestration. The control plane must allocate slices, assign learning functions, and monitor resource utilization in real time [12]. Designing such an orchestration engine that bridges multiple administrative domains and operates under diverse service-level agreements constitutes a significant systems-engineering undertaking.

## 1.2 Motivation

The confluence of emerging 6G capabilities and maturing DL techniques provides an unprecedented opportunity to reshape vehicular intelligence. NS, standardized in 5G, offers the means to isolate and tailor virtual networks for distinct IoV services, ensuring predictable quality of service (QoS) and efficient resource usage [13]. By embedding learning workflows within slices, rather than treating them as external applications, network operators can orchestrate compute and communication resources holistically.

Moreover, NTN promises to complement TNs by extending coverage, reducing reliance on cell towers, and providing additional aggregation points for model updates. Employing satellites or HAPs as FL coordinators can alleviate bottlenecks in rural or infrastructure-scarce regions, thereby democratizing the benefits of Intelligent Transportation Systems (ITS) [14].

From the algorithmic perspective, FL ensures data locality by keeping raw samples on vehicles, thus aligning with privacy regulations. SL further segments model execution to sidestep resource constraints, while TL leverages large-scale pretrained backbones to accelerate convergence on vehicular tasks. Integrating these paradigms within a slice-aware architecture promises to meet the stringent latency, reliability, and privacy requirements of 6G-IoV applications [15].

Finally, adaptive control systems powered by DRL can optimize resource and bitrate allocation for latency-sensitive services such as real-time video streaming and cooperative sensing. By learning policies that adapt to network dynamics and mobility patterns, these agents can enhance user experience, improve spectrum utilization, and maintain robust performance under uncertainty [16].

### 1.3 Contributions

The main contributions of this thesis are as follows.

- We introduce DLaaS, a novel network-sliced framework that virtualizes FL, split-model training, and Knowledge Distillation (KD) workflows within isolated network slices. DLaaS dynamically orchestrates learning functions across edge, cloud, and non-terrestrial infrastructures to satisfy diverse latency, reliability, and privacy requirements in IoV environments.
- We develop FSTL and its hierarchical generalization (GFSTL), which partition deep model architectures across vehicles, RSUs, and aerial coordinators. By leveraging pretrained backbones and adaptive split-point selection, FSTL and GFSTL reduce communication overhead and accelerate convergence compared to classical FL and SL approaches in multilayer ITS scenarios.



- We demonstrate the practical viability of various DL algorithms on resource-constrained platforms, specifically Raspberry Pi and Odroid devices, by incorporating warm-start initialization and adaptive client selection policies. Our prototype evaluations reveal significant improvements in convergence speed, model accuracy, and energy efficiency under realistic hardware heterogeneity and thermal constraints.
- We design and evaluate a multi-agent DRL controller within an Edge-DASH architecture for adaptive video streaming. The proposed agents jointly optimize bitrate allocation, caching strategies, and user-to-server assignments, resulting in enhanced playback stability and cache hit rates under dynamic network and vehicular mobility conditions.

## 1.4 Organization

The remainder of this thesis is structured as follows. **Chapter 2** presents the DLaaS architecture in detail, including its functional components, slice orchestration algorithms, and performance evaluation in simulated IoV scenarios. **Chapter 3** describes the FSTL and GFSTL frameworks, offering theoretical analysis, algorithmic designs, and simulation results in multilayer ITS environments. **Chapter 4** discusses the implementation of DL algorithms on Raspberry Pi, Odroid hardware, and Virtual Machines (VMs), detailing experimental setups, performance metrics, and energy-profiling outcomes. **Chapter 5** introduces DRL-based adaptive video streaming over Edge-DASH, encompassing system modeling, learning algorithm design, and numerical evaluations. Finally, **Chapter 6** synthesizes the findings, reflects on the broader implications, and outlines directions for future research in distributed ML for 6G-enabled VNs.

# Chapter 2

## Network-Sliced Distributed Learning for 6G IoV

Some content of this chapter is based on the following article [\[17\]](#):

*David Naseh, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Network sliced Distributed Learning-as-a-Service for Internet of Vehicles applications in 6G non-terrestrial network scenarios." Journal of Sensor and Actuator Networks 13, no. 1 (2024): 14.*

### 2.1 Introduction

Recently, Machine Learning (ML) techniques, especially those belonging to the Distributed Learning (DL) class, have gained huge popularity in dynamic wireless scenarios such as the Internet of Vehicles (IoV) with their added advantages in terms of learning efficiency, reliability, and data security [\[18\]](#). With this, various DL methods,

such as Federated Learning (FL), Multi-Agent Learning, and Collaborative Learning, are considered in vehicular domains [19]. Additionally, various ML tools and techniques have been considered to form suitable DL methods, such as multi-agent FL, DL with model split, DL with meta-Learning, and DL with swarm learning. In this way, a rich ecosystem of DL methods with specific characteristics, performance, and demand is formed and made available to serve users [20].

From a networking point of view, several new advances have recently been introduced, especially with the innovations of 5G and B5G technologies. Different computing paradigms, such as Edge/Cloud Computing, have been introduced to implement new services and applications with better performance [21]. Technologies, such as network softwarization through Network Function Virtualization (NFV), Software Defined Networking (SDN), and Network Slicing (NS), have revolutionized the networking process and opened the doors to a multitude of applications and services with different demands and additional flexibility [22]. Furthermore, distributed computing and communication technologies, such as the edge-to-cloud continuum [23], and joint Terrestrial and Non-Terrestrial Networks (T/NTN) [24], have gained huge popularity in terms of capacity, coverage, and reliability for serving end users.

In the realm of advanced 6G technologies for NTN and satellite-terrestrial integrated networks, the contributions are notable as follows. Ref. [25] delves into secrecy-energy-efficient hybrid beamforming, proposing robust schemes for single and multiple earth stations. The authors in [26] focused on a destructive beamforming design, introducing low-complexity schemes for known and unknown malicious Reconfigurable Intelligent Surfaces (RIS). Ref. [27] addresses joint beamforming for hybrid satellite-terrestrial relay networks, optimizing power while ensuring user rate requirements. Lastly, the authors in [28] proposed a multilayer RIS-assisted secure

integrated terrestrial-aerial network architecture, maximizing system energy efficiency and defending against attacks.

Shifting focus to broader aspects of wireless communication, some of the latest key technologies that contribute to the applications of different 6G techniques in Intelligent Transportation Systems (ITS) and IoV, and the vision for 6G-enabled smart cities are as follows. Ref. [29] emphasizes cooperation between cognitive users for better cognitive radio network performance. The authors of [30] explored the role of 3rd Generation Partnership Project (3GPP) in the evolution of cellular communication, highlighting the sharing of resources in the context of the IoV. Ref. [31] delves into interference in cognitive radio networks, discussing terminologies and mitigation techniques. Finally, Ref. [32] provides a visionary perspective on 6G-enabled Internet of Things (IoT) networks for sustainable smart cities, incorporating Artificial Intelligence (AI), ML, and novel architectures.

These technologies, along with different ML methods, are creating a new paradigm, known as Edge Intelligence, to enable distributed near-user intelligent wireless networks in different domains [33]. However, there are still some gaps between the potential of these innovative technologies and their possible uses to create a safe, reliable, and intelligent vehicular system. When focusing on the IoV scenario, users demand an increased number of services, each tailored to specific requirements [34]. For example, autonomous driving may require huge data processing and low latency, while infotainment applications may require ultra-broadband connections. The need to cope with several applications has a large impact on the need to create an intelligence-at-the-edge environment that can adapt to varying demands and different requirements proactively. To this end, NS is a perfect tool, enabling the possibility of logically managing network resources from both the communication

and processing points of view, thus the possibility of providing several services in a flexible way at the same time [35].

In this environment, we focus on the proposed framework, where different DL algorithms, each tailored to specific requirements, can be deployed, allowing the possibility of managing heterogeneous intelligent services simultaneously. Although in the landscape of advanced 6G technologies for integrated T/NTN the literature highlights advancements, a comprehensive integration of DL techniques with NS in the IoV context is underexplored. Our motivation stems from the identified gaps in recent works, leading us to propose a DL-as-a-Service (DLaaS) framework for vehicular environments within 6G NTN scenarios. The existing literature focuses on specific aspects, but a holistic framework that seamlessly integrates DL, NS, and distributed computing/communication methods for proactive delivery of intelligent services is missing.

Our main contributions include:

- **Comprehensive Analysis:** We conduct an in-depth exploration of key technological innovations and various DL methods, laying the groundwork for subsequent developments.
- **DLaaS Framework:** We introduce an innovative DLaaS framework, representing a paradigm shift that seamlessly integrates DL, NS, and distributed computing/communication methods.
- **Adaptation for IoV:** We elucidate the adaptation of the framework specifically for IoV applications, revealing its potential in shaping the future of intelligent Vehicular Networks (VNs).

- **Case Study and Performance Analysis:** We provide a detailed case study and performance analysis, offering empirical evidence of the efficacy and practical implications of DLaaS in real-world scenarios.

Finally, we will see that our proposed DLaaS approach offers advantages in terms of enhanced performance, flexibility, scalability, and intelligence, thus addressing the identified gaps and contributing to the evolution of intelligent vehicular communication systems.

To elaborate more on our proposal, Section 2.2 delves into a meticulous analysis of key technological innovations alongside various DL methods, laying the foundation for our subsequent developments. Section 2.3 introduce the innovative DLaaS framework. This framework represents a paradigm shift that seamlessly integrates DL, NS, and distributed computing/communication methods. Integration empowers the proactive delivery of intelligent services to users, fostering a dynamic environment that caters to diverse and evolving requirements. Furthermore, we elucidate the adaptation of this framework specifically for IoV applications, unveiling its potential in shaping the future of intelligent VNs. This section also provides a nuanced examination of both the challenges and advantages associated with the DLaaS framework. The culmination of our exploration is presented in Section 2.4, where we present a detailed case study accompanied by a performance analysis. This case study serves as a tangible demonstration that illustrates the advantages that the DLaaS approach offers in terms of flexibility, performance enhancement, and infusion of added intelligence into vehicular communication systems. Through this detailed study, we not only contribute to the theoretical framework but also provide empirical evidence of the efficacy and practical implications of DLaaS in real-world scenarios.

## 2.2 IoV Distributed Intelligence

### 2.2.1 Edge/Cloud Computing

Cloud-based infrastructures with abundant resources to meet end-user requirements were one of the popular solutions in the early part of the last decade. However, over time, several issues occurred when considering cloud-based infrastructures to compute end-user data. Longer transmission distances and the corresponding communication costs, data security threats, and backhaul congestion were among the main issues that reduced the impact of cloud technology over time. Furthermore, with the new advanced technologies, such as 5G, new services with limited latency and high data rate requirements were enabled, further placing additional burdens on cloud facilities. However, with new technologies, the end devices also evolved to have powerful on-board computation capabilities, and with that, abundant computation power distributed over the network area was added. This gives birth to Fog/Edge computing technologies, bringing cloud computing facilities closer to end users [36]. Over the years, EC has achieved great success in terms of providing end users with high-quality services with limited latency [37].

Although EC has solved some of the cloud computing problems, its size limitations place additional restrictions on the computation, communication, and storage resources of edge facilities [38]. In recent times, as we move toward 6G, it has been seen that EC facilities are overwhelmed, and new solutions are required to fulfill the demands of new services. With this, different distributed networking infrastructures, such as the edge-cloud continuum and the Integrated T/NTN infrastructure, are considered and expected to play an essential role in the near future IoV scenarios [24, 39]. These distributed networking infrastructures can have large sets

of diverse networking nodes with on-board computing/storage resources. Furthermore, with different communication technologies, these devices can communicate effectively with each other and with end users. Thus, a huge amount of distributed computing and communication power is available for the implementation of DL methods in such distributed networking infrastructures.

### **2.2.2 Network Slicing**

Network softwarization is an important emerging trend in 5G and B5G-based networking systems aimed at creating a flexible network architecture with a reduced time to market for new services. NS is one of the major enabling technologies of the network softwarization realm allowing one to support diverse sets of services. NS has been introduced in the context of 5G, allowing mobile operators to create and customize their networks to provide optimized solutions for different market scenarios with diverse requirements [22]. Among others, Automation, Isolation, Customization, Elasticity, Programmability, End-to-End, and Hierarchical abstraction are the main principles of NS technology. NS provides dynamic resource management by enabling efficient resource sharing by considering various key performance indicators (KPIs) for each slice. NFV and SDN are two of the main technologies that enable NS over a common networking infrastructure. With NFV, network functions can be decoupled from proprietary hardware and run as software instances over virtualized environments, allowing them to overcome the lack of flexibility of traditional hardware-based network functions. On the other hand, SDN can help to create a fully softwarized wireless network by logically separating the data and control plane.

Recently, the vehicular community has shown great interest in NS technology for providing emerging services with complex structures to end users in a limited time [40].

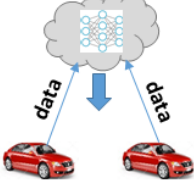
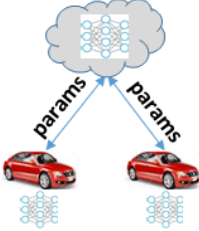


Thus, the implementation of vehicular services over a distributed VN through the deployment of several logical slices is gaining importance. Within this scenario, DL is a fundamental element required to support advanced IoV services. In the context of the DL ecosystem, a set of functions with different interdependencies is required to be executed. For example, in the case of centralized FL, functions such as data acquisition, data cleaning, hyperparameter settings, learning technique selection, data training, the transmission of learning data from devices to servers, data processing at the server, the averaging process performed by the server, and broadcasting of global model data from the server to devices are required to be implemented for completing one single learning iteration. Different forms of FL with advanced learning tools and technologies (i.e., FL with Transfer Learning (TL)) can require additional sets of functions. Several of these functions can be implemented as virtualized learning functions on network infrastructures. With the availability of distributed computing and communication infrastructures, such as the edge-cloud continuum and integrated T/NTN, along with virtualization technologies, these learning functions can be implemented at different locations based on their characteristics and requirements.

### 2.2.3 Overview of Distributed Learning Methods

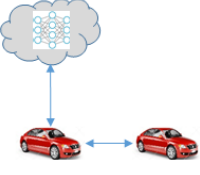

In this part, we will discuss the fundamental structures and the distinctions between Centralized Learning (CL) and various DL approaches, which are illustrated in Table 2.1. For more explanation, see [20, 41].

TABLE 2.1: Advantages, challenges, conditions and KPIs for different learning paradigms.

| Learning Method           | Advantages   | Challenges  | Conditions   | KPIs  |
|---------------------------|--|---|--|---|
| Centralized Learning (CL) |   |   |  |   |
|                           | <ul style="list-style-type: none"> <li>• Able to train complex ML models</li> <li>• Better performance for non-convex applications</li> <li>• Less impact of communication link imperfections</li> </ul> | <ul style="list-style-type: none"> <li>• Sharing private data with a centralized controller</li> <li>• High transmission overhead</li> <li>• Edge devices' limited resources &amp; energy</li> <li>• Additional delays to a central server</li> </ul> | <ul style="list-style-type: none"> <li>• Devices' willingness to share data</li> <li>• Connectivity for data transmission</li> <li>• Relaxed latency requirements</li> <li>• Complex training processes</li> </ul> | <ul style="list-style-type: none"> <li>• Delay: High</li> <li>• Privacy: Low</li> <li>• Mobility: Low</li> <li>• Processing: High</li> </ul>        |
| Federated Learning (FL)   |   |   |  |   |
|                           | <ul style="list-style-type: none"> <li>• Elevated sensitive-data privacy</li> <li>• Distributed model learning</li> <li>• Edge/device-level training</li> </ul>  | <ul style="list-style-type: none"> <li>• Data heterogeneity across nodes</li> <li>• Single point of failure</li> <li>• Unreliable communications</li> <li>• Large parameter-exchange overhead</li> </ul>  | <ul style="list-style-type: none"> <li>• Reliable node-server links</li> <li>• Low-complexity models</li> <li>• Edge devices able to train locally</li> </ul>  | <ul style="list-style-type: none"> <li>• Delay: Medium</li> <li>• Privacy: Medium</li> <li>• Mobility: Low</li> <li>• Processing: Medium</li> </ul> |

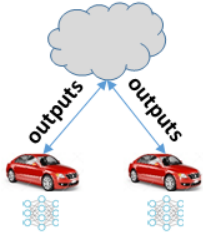
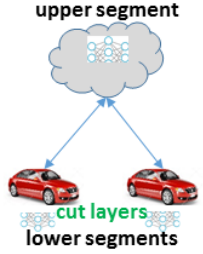
*Continued on next page*

*Table 2.1 continued from previous page*

| Learning Method                              | Advantages  | Challenges  | Conditions   | KPIs   |
|--|---|---|--|--|
| Collaborative<br>Federated Learning<br>(CFL) |  <ul style="list-style-type: none"> <li>• Elevated data privacy</li> <li>• More training data</li> <li>• Scalable to large systems</li> </ul>                    | <ul style="list-style-type: none"> <li>• Transmission inefficiency</li> <li>• Lower convergence vs. FL</li> <li>• Variable device accuracy</li> </ul> | <ul style="list-style-type: none"> <li>• Reliable inter-device links</li> <li>• Devices can train/aggregate</li> </ul>   | <ul style="list-style-type: none"> <li>• Delay: Low</li> <li>• Privacy: Low</li> <li>• Mobility: Medium</li> <li>• Processing: Medium</li> </ul> |
| Group ADMM<br>(GADMM)                        |  <ul style="list-style-type: none"> <li>• Only half devices compete per round</li> <li>• Neighbour-only communication</li> <li>• Reduced energy use</li> </ul> | <ul style="list-style-type: none"> <li>• High payload per communication</li> <li>• Limited scalability</li> <li>• DNN exchange hampered</li> </ul>    | <ul style="list-style-type: none"> <li>• Stable links in dynamic environments</li> <li>• Limited node coverage</li> <li>• Ample comms &amp; compute</li> </ul> | <ul style="list-style-type: none"> <li>• Delay: Low</li> <li>• Privacy: Low</li> <li>• Mobility: High</li> <li>• Processing: Low</li> </ul>      |


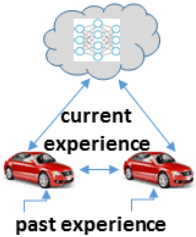
*Continued on next page*

Table 2.1 continued from previous page

| Learning Method                | Advantages   | Challenges   | Conditions   | KPIs  |
|--------------------------------|--|--|--|---|
| Federated<br>Distillation (FD) |  <ul style="list-style-type: none"> <li>• Only model-output exchange</li> <li>• Suited to limited wireless</li> <li>• Handles heterogeneous models</li> <li>• RL via neighbour averaging</li> </ul> | Vulnerable to non-IID data   | <ul style="list-style-type: none"> <li>• Same output task on all devices</li> <li>• IID data required</li> </ul> | <ul style="list-style-type: none"> <li>• Delay: Medium</li> <li>• Privacy: Medium</li> <li>• Mobility: Low</li> <li>• Processing: Medium</li> </ul> |
| Split Learning (SL)            |  <ul style="list-style-type: none"> <li>• Splits DNN layers across devices</li> <li>• Robust to non-IID data</li> </ul>   | <ul style="list-style-type: none"> <li>• Frequent forward/back exchanges</li> <li>• Cut-point choice affects cost</li> </ul> | <ul style="list-style-type: none"> <li>• Stable vehicular links</li> <li>• Optimal layer-cut design</li> </ul>   | <ul style="list-style-type: none"> <li>• Delay: Medium</li> <li>• Privacy: High</li> <li>• Mobility: Low</li> <li>• Processing: High</li> </ul>     |

*Continued on next page*

*Table 2.1 continued from previous page*

| Learning Method           | Advantages   | Challenges  | Conditions  | KPIs  |
|---------------------------|--|---|---|---|
| Multiagent RL<br>(MARL)   |  <ul style="list-style-type: none"> <li>• Exploration + exploitation</li> <li>• Agents learn &amp; adapt</li> </ul>   | <ul style="list-style-type: none"> <li>• No guaranteed policy equilibrium</li> <li>• Extra comms for convergence</li> </ul>   | <ul style="list-style-type: none"> <li>• Stable user-environment links</li> <li>• Equity convergence guaranteed</li> </ul>  | <ul style="list-style-type: none"> <li>• Delay: High</li> <li>• Privacy: Low</li> <li>• Mobility: Medium</li> <li>• Processing: High</li> </ul> |
| Transfer Learning<br>(TL) |  <ul style="list-style-type: none"> <li>• Better data quality &amp; quantity</li> <li>• Faster convergence</li> <li>• Lower compute &amp; comms</li> <li>• Privacy preserved</li> </ul> | <ul style="list-style-type: none"> <li>• Choosing source task &amp; layers</li> <li>• Deciding transfer parameters</li> </ul> | <ul style="list-style-type: none"> <li>• Past experience available</li> <li>• Online data &amp; memory available</li> </ul> | <ul style="list-style-type: none"> <li>• Delay: Low</li> <li>• Privacy: Medium</li> <li>• Mobility: Low</li> <li>• Processing: High</li> </ul>  |

**Centralized Learning (CL):** In the case of CL, a set of distributed wireless nodes (e.g., vehicles) needs to communicate their collected local datasets over an unreliable communication channel to the centralized entities (i.e., base stations, clouds, etc.) for the ML model training. This process often results in higher data transmission costs, training latency, data security issues, etc. In the case of resource-constrained

nodes, such as the IoV scenario, these issues become more critical. For this reason, DL is preferred to latency-critical VNs to perform various learning tasks.

In the following, the main DL methods are discussed in terms of characteristics, requirements and suitability to solve vehicular problems [42], where the main difference in DL versus CL is due to the fact that the ML model is trained at different locations and proper information exchange is performed among nodes.

**Federated Learning (FL):** FL is one of the most widely used DL techniques, where wireless nodes perform distributed training operations with the help of a centralized parameter server. The FL process includes two main steps. First, devices with their local datasets perform the model training operation and communicate parameter updates to a server without the need to share their sensitive raw data. In the second step, the parameter server collects and aggregates the model updates from all devices to create a global learning model that has the benefit of aggregated training experiences from all devices. This global model is then used again by the nodes in the next FL iteration, allowing them to learn from the other devices' training experiences. It is especially helpful in IoV, where FL enables collaborative model training while respecting individual data ownership and security [43]. In some situations, the traditional centralized FL is not convenient and further optimization is needed. For this reason, in the recent past, different forms of FL have been proposed, especially to optimize FL performance according to learning environments [44]. Hierarchical FL with FL process distributed over several layers of edge devices [45], Federated Distillation (FD) [46], FL with TL and FL with SL [47] are examples of updated FL-based techniques.

**Collaborative Federated Learning (CFL):** In reality, devices may not be able to connect to the central node due to energy constraints or possibly high transmission latency. To address this issue and make FL more accessible in real-world scenarios,

the concept of CFL has been introduced, which allows vehicles to participate in FL without communicating with the central unit [48]. Devices that cannot connect directly to the central node can interact with adjacent vehicles. In this paradigm, each device can be connected to its nearest vehicle. This learning method is also trained iteratively. First, each device sends its trained local FL model to its connected devices or to the central node. The central node then produces the global FL model and sends it to the corresponding devices. Finally, each device changes its local FL model depending on the FL parameters received from other devices or the BS. In FL, each device may train its local FL model using gradient descent (GD) techniques, while the BS aggregates the local FL models. In CFL, however, each device must both aggregate the local FL models received from other devices and train its own local FL model.

With the presence of high-quality computation hardware such as multi-core Central Processing Units (CPUs), Graphical Processing Units (GPUs), and Tensor Processing Units (TPUs), vehicular nodes can themselves train the learning models without the need for parameter servers. In some cases, with reduced mobility, and through V2X technology, vehicular nodes can collaborate to solve learning tasks. Such a CFL approach can be highly efficient in terms of training. Without the presence of a third party in the learning process, this can also further strengthen vehicular data security and improve model convergence and efficiency, making it applicable to IoV scenarios with diverse and dynamic data sources [49].

**Federated Distillation (FD):** FD leverages outputs from models rather than the parameters in FL. Since the output dimensions are significantly smaller than the model sizes, it is much more communication efficient [46]. For example, each device in a classification task performs local iterations while saving the average model output for each class. These local average outputs, which aggregate and average

the local average output among agents in each class, are sent to the central node regularly. Each device downloads the results that constitute the resultant global average. Finally, each agent runs local iterations with their loss function in addition to a regularizer that measures the difference between its prediction output of a training sample and the global average output for the given class of the sample, which is called knowledge distillation (KD), to translate the downloaded global knowledge into local models. FD can also be beneficial in IoV to improve model performance in a network of vehicles [50].

**Group alternate direction method of multipliers (GADMM):** The FL central node may not be able to communicate with remote edge devices. It can also be vulnerable to failure or act as a single point of attack. To this aim, GADMM intends to provide DL without a central entity by using the alternate direction method of multipliers (ADMM) technique and interacting exclusively with surrounding devices by splitting the devices into head and tail groups. Only two devices from the tail/-head group are selected and create a chain, with each device from the head or tail group exchanging variables. With GADMM, only half of the agents compete for the restricted bandwidth during each communication cycle. Furthermore, by restricting communication to two nearby agents, the communication energy may be greatly reduced [51]. In IoV, GADMM can be used for collaborative decision making, traffic flow optimization, or other distributed tasks, such as energy-efficient resource allocation [52].

### **Split Learning (SL):**

SL is a technique that allows resource-limited wireless devices to train complex models such as DNNs. During the DNN training process, the model can be split vertically or horizontally, allowing multiple nodes to train a portion of the model with limited data samples and training latency. SL combined with different forms of DL



can be useful in dynamic vehicular settings for producing reliable complex learning models. Moreover, since SL does not exchange raw data, data privacy is somewhat maintained [47]. This approach is particularly relevant for IoV applications where model inference can occur locally on vehicles.

**Multi-Agent Reinforcement Learning (MARL):** When environmental dynamics influence agents' decisions, they must learn about these dynamics and adjust their methods based on experience gained via agent-to-environment and agent-to-agent interactions. In this regard, Reinforcement Learning (RL) with exploration and exploitation abilities is critical. Exploring in RL allows agents to understand the dependencies of their decisions on the environment and other agents (policy) and on the consequences (value), which may then be used to improve long-term rewards. Even in single-agent instances, the data necessary to understand policy and value might be dispersed over several agents acting as helpers. FL, FD, and GADMM can improve learning policies and value over distributed helpers despite communication and privacy constraints. Within the MARL paradigm, the interactions of several agents in the same environment while making decisions based on local observations are investigated [53]. MARL is classified into centralized/decentralized and cooperative/competitive frameworks based on the presence of a central controller and the sorts of interactions. Centralized MARL frameworks assume a central controller that learns decision-making rules by gathering all agents' experiences, which include observed states, actions taken, and rewards received. Exchanging such information may use a significant amount of communication and memory resources, while jeopardizing data privacy. Decentralized MARL without a central controller does not have these disadvantages, but it does not ensure individual agents' convergence to equilibrium policies, even in cooperative MARL where all workers aim for the same

objective. In IoV, MARL can be applied to edge caching [54], cooperative navigation [55], traffic optimization [56], and other scenarios in which vehicles interact with each other.

### **Transfer Learning (TL):**

With the involved dynamicity, resource limitations, and latency constraints, performing full-scale model training is not always feasible in vehicular environments. To this end, TL can be very useful for performing model training. In the case of TL, learning agents can utilize past learning experiences through knowledge transfer (KT) to perform new learning tasks. TL approaches can increase the convergence rate, minimize reliance on labeled data, and improve the robustness of ML techniques in different vehicle settings [57]. There are various forms of TL based on KT strategies. For example, the experiences gathered in terms of learning data, e.g., data features and learning data scope, can be transferred for efficient learning of target tasks. On the other hand, knowledge depending on a trained model, for example, the structure and parameters of the model, can also be shared with a target task to improve training performance [58].

The advantages, challenges, conditions, and KPIs of these fundamental structures are illustrated in Table 2.1 in order to distinguish them in different applications. The main KPIs, which are used later in the case study section, including delay, preserved privacy, mobility of handled vehicles, and gained processing capability, are ranked in the last column of the table.

## 2.2.4 Intelligence at the Edge for IoV

### 2.2.4.1 Applications

The fusion of IoT and AI, known as AIoT, is transforming IoV to improve road safety, efficiency, and mitigate traffic issues [59]. The IoV landscape comprises three main categories: AD, Safe Driving Monitoring Systems, and Cooperative Vehicle Infrastructure Systems (CVIS).

**Autonomous Driving:** To address the challenges of massive data generation (i.e., 4000 TB per day) and the need for real-time decision making, EC emerges as a viable solution [60]. AD signifies a shift toward intelligent vehicles capable of AI-driven decision making. EC, exemplified by vehicles such as HydraOne [61] and HydraMini [62], addresses the challenges of massive data generation and allows real-time decision making in critical scenarios. Furthermore, edge-based systems such as EdgeDrive improve safety through real-time Advanced Driver Assistance Systems (ADAS) applications [63].

**Safe Driving Monitoring Systems:** Driver monitoring systems, crucial for safe driving, combat issues such as drowsiness. In [64], a Raspberry Pi 3-based system was implemented that uses a DL algorithm for real-time alerts by analyzing facial features captured in both day- and night-drive scenarios.

**Cooperative Vehicle Infrastructure Systems:** CVISs establish real-time road information networks by connecting vehicles, pedestrians, and infrastructure. Using distributed infrastructure, including vehicles, base stations (BSs), and roadside units (RSUs), EC reduces transmission delays for timely communication. In [65], the authors proposed a You Only Look Once (YOLO) DL model for car accident

detection (YOLO-CA) system that uses 5G networks detects accidents promptly, using the CAD-CVIS dataset for improved accuracy.

In conclusion, the integration of edge intelligence into IoV brings notable advances in AD, safe driving monitoring, and CVIS, addressing data challenges and fostering cooperative systems for improved road safety and traffic management. The reviewed literature emphasizes the transformative impact of AI at the edge in the IoV landscape.

#### 2.2.4.2 Aspects and Advantages

Intelligence at the edge in the context of the IoV refers to the deployment of computational and analytical capabilities directly within the vehicles or at the network's edge, rather than relying solely on centralized cloud-based processing. In IoV applications, as mentioned above, the need for real-time decision-making, decreased latency, increased efficiency, and improved privacy is what motivates this strategy. Here are key aspects and advantages related to intelligence-at-the-edge in IoV:

1. **Real-time Decision-Making:** By embedding intelligence at the edge, vehicles can make local, real-time decisions without relying on a centralized cloud server. This is critical for applications such as emergency braking and collision avoidance that require quick reactions [66].
2. **Reduced Latency:** EC minimizes the delay in processing the data, since computations occur closer to the source of the data. This is particularly crucial in the IoV, where accurate and timely responses to dynamic traffic conditions depend on low-latency communication [67].

3. **Bandwidth Efficiency:** Processing data at the edge reduces the need to transmit large amounts of raw data to a central server for analysis. Instead, only relevant or summarized information can be sent, optimizing bandwidth usage in IoV networks [68].
4. **Enhanced Privacy and Security:** Edge intelligence allows data processing to occur locally, addressing concerns related to privacy and security. Sensitive information can be processed within the vehicle, minimizing the exposure of personal data to external networks [69].
5. **Distributed Computing:** EC in IoV involves a distributed computing paradigm where intelligence is distributed across vehicles and road infrastructure. This decentralized approach enables collaborative decision-making and more efficient utilization of resources [70].
6. **Scalability:** EC supports scalability in IoV applications. As the number of connected vehicles increases, edge devices can handle processing tasks locally, preventing bottlenecks on centralized cloud servers [71].
7. **Adaptive learning:** Intelligent edge devices can employ ML algorithms to adapt and improve their performance based on the data they process. This adaptability is valuable in IoV scenarios where traffic patterns and conditions can change dynamically [72].
8. **Offline operation:** Edge intelligence allows vehicles to perform certain tasks even when not connected to the central network. This offline operation is beneficial in scenarios where network connectivity may be intermittent or unavailable [73].

In summary, deploying intelligence at the edge in IoV applications offers the above-mentioned advantages. This approach aligns with the dynamic and distributed

nature of IoV, which contributes to more efficient and responsive connected vehicle systems.

## 2.3 Network Sliced Distributed Learning

As introduced, DL is a promising technology for designing intelligent vehicular networking systems. However, mapping different DL functions for different IoV services and requirements represents a challenge. We propose here a DLaaS concept that allows the implementation of multiple DL operations over the distributed VN through the deployment of specific learning slices, where each DL method can be seen as the composition of multiple virtual functions.

### 2.3.1 End-to-End Functional Decomposition of DL

As described in the previous section, different DL methods can be characterized by different sets of functions that must be implemented in distributed networks to have the proper benefits. Thus, each DL approach can be implemented as a set of functions coordinated through a chain characterized by functional dependencies. For this, an adequate functional decomposition providing a set of typical DL functions is needed. After having discussed the different learning techniques in distributed environments, here we propose a set of possible learning functions that are needed for implementing various DL methods:

- **Data Acquisition Function (DAF):** Generally, distributed training operations involve several learning devices collaboratively performing the learning process. In the case of vehicular scenarios, this can be geographically distributed sets of vehicles moving across road networks. For the case of DL,

learning devices need their own datasets to perform the training process, which can be collected through a data acquisition mechanism that involves a set of sensory nodes, processing devices, and data collection devices. The process that allows the composition of the learning dataset can be defined through a typical DAF. Note that such functions can only be implemented on nodes/devices with typical hardware settings. With new vehicular nodes equipped with several sensory nodes, they can collect large amounts of data samples over time through DAF that can be exploited for a successful implementation of DL.

- **Data Preprocessing Function (DPrF):** In general, the learning data acquired through DAF can be in different formats, e.g., texts, images, videos, etc. Based on the learning tasks, the selected learning method, and their requirements, these data need to be pre-processed in a typical form. This can be achieved through learning data preprocessing methods implemented through a DPrF. This function can have methods for data cleaning, data dimensionality reduction, data normalization, etc. DPrF function can help reduce the overall size of the original datasets, and thus reduce the communication overhead for some typical DL methods where data parallelization techniques involving learning data transfer are needed. Thus, preferably, this function needs to be implemented alongside the DAF function to avoid possible communication overheads.
- **Distributed Learning Function (DLF):** In DL frameworks, the learning process can be performed on different nodes, e.g., end devices or edge nodes, according to the learning frameworks adopted. The end devices can do the learning process themselves for some simplified learning tasks. In some cases, collaborative learning frameworks can be adopted for complex learning models such as DNN, allowing different devices to collaboratively train the models

(e.g., through different model split techniques). In another case, a data parallelism approach can be adopted, allowing struggling end devices with limited computational resources to send their data to the nearby devices/edge nodes to complete the training process in time. Therefore, a DLF is needed that adopts the selected learning strategy for the successful implementation of DL. Typical learning steps, such as learning model selection, hyperparameter settings, stochastic gradient descent (SGD), and backpropagation, can be part of a holistic DLF that can be implemented on distributed nodes.

- **Data Post Processing function (DPsF):** In a typical DL process, after performing the learning steps through DLF, the parameter updates must be sent to the parameter server or other learning nodes based on the adopted learning strategy. Often, data processing is needed to avoid communication overhead, limit data security risks, and add the appropriate weighting coefficients to the learning process results. This method can be implemented through a DPsF that processes the learned data before its transmission to the outside world.
- **Data Collection Function (DCF):** In each DL cycle, parameter servers are required to collect learning updates from the devices and create a global update that can be used for the next round of communication. The data received by the servers may have additional information, encryption, noise, etc., and are required to be processed before taking into account the global model update. The DCF includes the steps to collect data from learning devices and prepare them for the global update to be performed.
- **Global Model Update Function (GMUF):** The GMUF performs the updates of the learning model based on learning data. The DCF function results are further processed with some mechanisms, i.e., the averaging process for creating a global model. These model parameters are sent back to the devices



or upper layers for further processing. GMUF function can have methods for generating, pre-processing, and transmitting global model parameters over different distributed nodes.

- **Distributed Model Inference Function (DMIF):** Model inference is an important step that must be considered for the successful implementation of AI applications based on DL. End users can adopt various forms of inference mechanism for the successful implementation of DL applications based on resource availability and application requirements. These methods and processes can be included in the DMIF. Based on resource availability and application performance requirements, different model inference strategies can be adopted. If a model in question is simple and requires limited computations, inference can be performed on the device itself, increasing the data security. However, in the case where the model requires a large number of parameters with a large computation cost, inference operations can be performed at edge or cloud layers. In some cases, joint strategies (e.g., device-edge, device-edge-cloud) can also be adopted with model-split operations. This creates different possible deployment options for the DMIF function.

This set of functions can be used to implement various DL methods in the IoV scenario using the NS principle, aiming to logically deploy multiple intelligent services at the same time.

### 2.3.2 DL-as-a-Service for IoV Applications

#### 2.3.2.1 Proposed Methods

The IoV scenario considered is implemented through an integrated T/NTN equipped with EC platforms. Different networking nodes, such as Road Side Units (RSUs), Low-Altitude Platforms (LAPs), High-Altitude Platforms (HAPs), and Satellite nodes, are distributed throughout the service area. The system is able to take advantage of different DL methods to cope with the requirements of heterogeneous users. Since the scenario considered includes a massive amount of computation, communication, and storage resources distributed over the ground, air, and space networks, these resources can be utilized to create an intelligent VN through proper deployment of required DL methods, where each network device is able to host the virtual functions enabling the different DL execution. In such a system, a slice-based approach is considered, where each slice is a logical entity that enables the interconnection of different functions to build a specific DL method.

Without loss of generality, in Figure 2.1, four DL methods implemented in the form of slices are represented. The first slice aims to deliver an FL service performed on different layers of edge devices. In particular, end devices have their datasets to perform the learning process. For this, the DAF, DPrF, and DLF functions are deployed on a VU layer to enable the learning process. The learned parameters are then transmitted to nearby edge nodes (i.e., RSUs) through DPsF, limiting communication costs. The RSU node then collects the data from the VUs and performs the aggregation operations, for which DCF and GMUF are placed over it. The GMUF results are then transferred to the upper layer, where appropriate functions are present. The second slice aims to deliver DL with collaborative learning frameworks. The learning part is performed collaboratively over the user devices,

and appropriate learning functions are placed over the VUs cloud. The upper layers are used to create a generalized global model by aggregating the local models. The third slice is for the case of SL, where a data parallelization technique is adapted to split the learning process over the device and edge layers. Thus, learning functions are implemented both on the device and on the edge layer. Additionally, the HAP layer is used to create a global model. For latency-critical applications, a TL-based DL slice can be considered, where past learning experiences are integrated into current learning cycles to limit the learning process costs. The learning process is distributed over different edge layers.

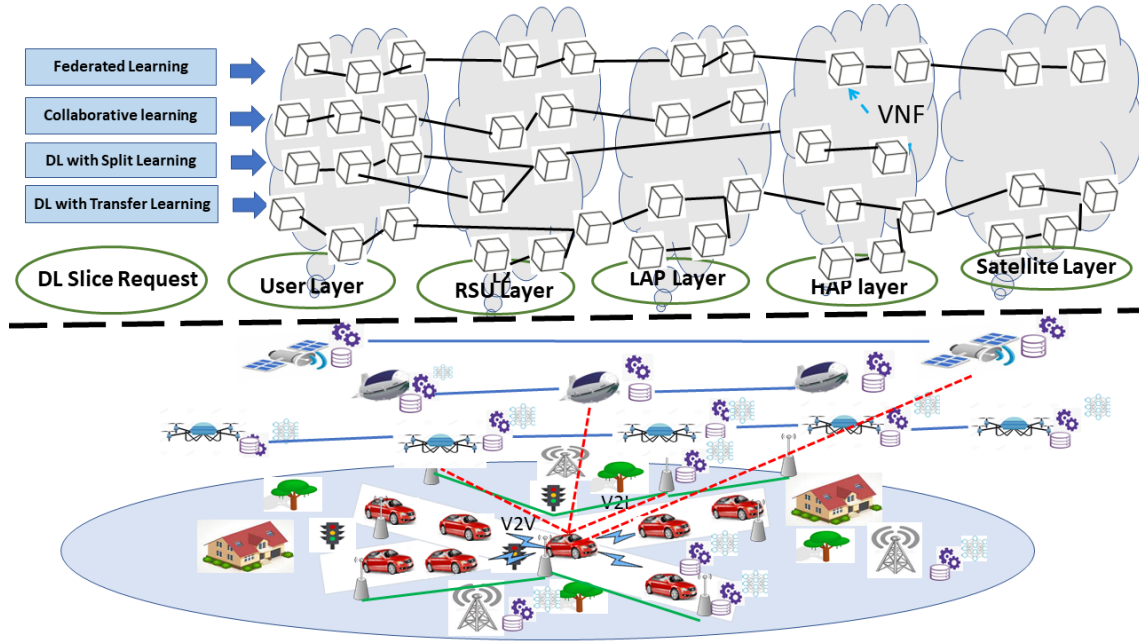


FIGURE 2.1: DLaaS over NS for IoV applications.

Figure 2.2 shows a more detailed view of the DLaaS concept, where the virtual learning functions of a single slice are reported. A multi-layer FL is considered representative, involving data collection and learning at the VU nodes, local/intermediate model updates collections and processing, i.e., averaging at the intermediate layers of RSUs, LAPs, HAPs, satellites, and model inference operation at VUs.

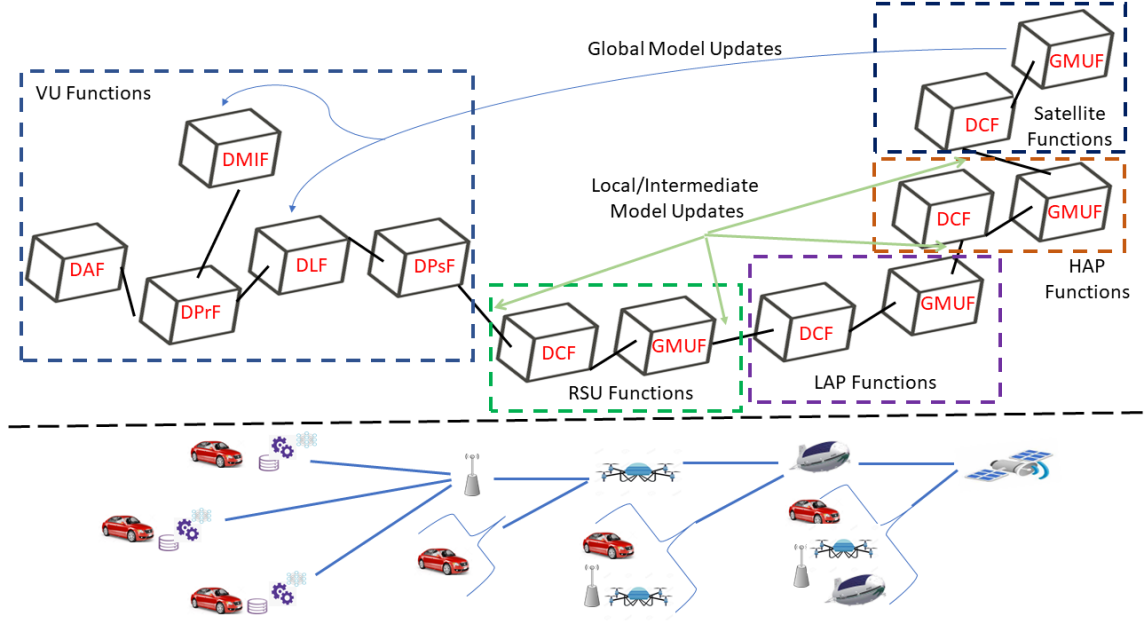


FIGURE 2.2: Multi-layer Federated Learning as a slice for IoV scenarios.

### 2.3.2.2 DLaaS Advantages

The proposed DLaaS approach introduces several advantages in terms of performance enhancement, flexibility, scalability, and intelligence. With the presence of multiple DL slices, the IoV system has *better performance* in terms of latency, energy costs, and overall learning performance. DLaaS allows different DL functions to be implemented on distributed platforms according to their specific requirements, local network conditions, and resource availability. This can improve performance by allowing several users/devices to participate in the learning process efficiently. Implementing various DL methods as slices on distributed computing platforms can provide additional *flexibility* in terms of resource sharing and slice function deployments. The NS approach also allows for better *scalability*, as it can enable a higher number of DL methods. This approach can also boost vehicle *intelligence* through the deployment of several possible DL slices simultaneously.

### 2.3.2.3 DLaaS Challenges

Though the proposed DLaaS method can have many advantages, several challenges must be considered while performing slice function deployments. These challenges can be based on application requirements, user-side demands, network restrictions, etc. The following key challenges should be considered in the proposed DLaaS method for proper benefits.

**Learning Function placement:** The DL functions can have specific requirements in terms of communication, computation, storage resources, and hardware dependencies. When placing the functions, it is also important to take into account the functional dependencies to avoid excess costs; for example, DPrF and DAF should be placed together to avoid the communication burden. The mobility of VUs, application requirements, and limited edge resources can add additional challenges. Thus, several of these issues can make the placement of the slice functions quite challenging, and improper placement of the functions can lead to an inefficient learning environment.

**Network Resource Allocations:** Learning slices can be data-intensive, computation-intensive, or communication-intensive based on their higher demands for storage, computing, and communication resources, respectively. Edge nodes can have a limited set of available resources that can change over time with various demands. Additionally, each node may host several learning functions from different DL slices. Additional constraints, such as mobility, unstable communication environments, and application requirements, can make the network resource allocation problem quite challenging.

**Multi-slice Implementations with Specific Demands:** DL slices are characterized by different function chains, requirements, etc., and implementing them over a

common infrastructure can be challenging. Each slice can have an impact on other slices' performance as a result of resource sharing. Multiple communication links enabled through a different set of slices can increase issues like noise, interference, etc.

**Security Threats:** Due to the presence of multiple DL slices with different user groups, the overall threat to data security can be elevated. Some slices/users can be more vulnerable to outside attacks and can end up impacting and compromising the security of other learning processes.

**High-Speed Distributed Computing and Communication Environment:**

The challenge of resource management in a high-speed distributed computing and communication environment is crucial for the efficient functioning of the DLaaS framework. As VNs operate in dynamic and high-speed environments, ensuring optimal resource allocation for fast-paced distributed computing becomes a significant challenge [74]. The need to manage resources such as computation, communication, and storage in real-time, considering the rapid movement of vehicles and the evolving nature of network conditions, adds complexity to the DLaaS implementation.

Thus, several of these challenges need to be properly addressed in order to have the additional benefits of DL as a Slice concept over a distributed vehicular networking environment.

### 2.3.3 DLaaS Architecture for IoV

Different IoV parameters, such as VUs' geographical positions, speed, edge node densities, application requirements, etc., can impact the VUs' demands for one of the available DL slices. Here, in Figure 2.3, we consider a case study of realistic IoV

scenarios with different slice demands. A centralized orchestrator/manager considers the scenario requirements where, thanks to specific DL slice descriptors, is able to deploy the Learning Functions to the different nodes. To this aim, proper descriptions of T/NTN layers are considered, where specific communications and computing capabilities are mapped. When assigning DL slices to the different VUs, they can be considered logically organized in clusters, where each cluster is characterized by specific applications, environmental conditions, and/or vehicular characteristics.

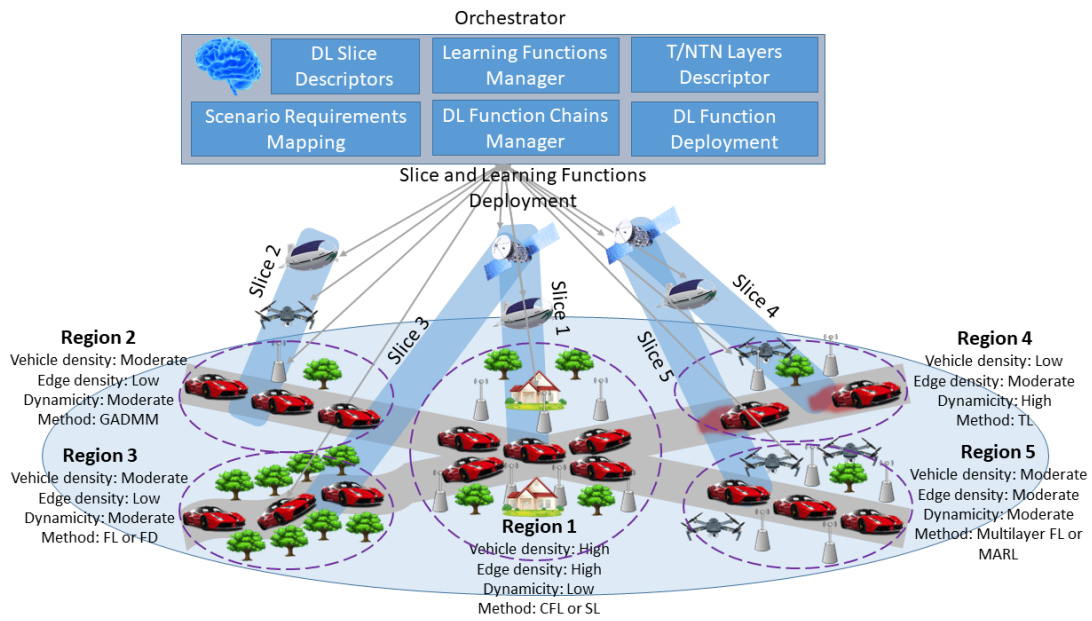


FIGURE 2.3: Slice and DL function deployment for regions with different IoV characteristics

As can be seen in Table 2.1, different DL algorithms can behave differently and, therefore, their selection depends on the selected IoV application, environmental conditions and characteristics of the IoV scenario, among others. For example, when using CFL, we must ensure that latency is not a critical factor, but we must also have a physical configuration that allows communication between nearby devices. To demonstrate the best approach, in Figure 2.3, we set up a configuration of vehicles,

RSUs, LAPs, HAPs, and satellites with different physical conditions based on the three qualitative vehicle speed levels, vehicle/edge density and scenario dynamicity.

Different application requirements, i.e., latency, privacy, and reliability, can dictate the choice of the DL slice along with the local environmental parameters. The orchestrator node must consider all these parameters and requirements before assigning the DL slices to the users. This approach can improve the performance of the DL in terms of model training costs, reliability, and user satisfaction.

In Figure 2.3, Region 1 represents an area where a large group of VUs is moving slowly (e.g., city centers, ring roads, traffic hotspots in the city). In addition, the area is covered with a moderate number of edge nodes on the ground and in space, providing distributed computing/communication services. In such scenarios, VUs collaboration needs to be exploited for efficient DL processes. To this end, collaborative FL or SL can be the best choice.

Region 2 highlights a road scenario with typical road environments (i.e., highways) having a limited number of edge nodes, to provide server VUs with steady speed. In such cases, the density of VUs will be limited, and collaborative learning might be challenging. Furthermore, with the limited number of edge nodes, advanced learning techniques, such as hierarchical FL, might not be feasible. However, with low edge node densities, only a few VUs might be able to connect to the servers. Such scenarios can be adequate to explain the benefits of GADMM-based DL.

Region 3 lacks terrestrial/aerial edge nodes and stable connections between users due to challenging physical conditions (i.e., dense forests, and hillsides). Such cases can motivate the use of FL for privacy-preserving applications.

Region 4 shows extremely good road conditions with high-speed VUs. Frequent handovers, limited training data, and unstable channel conditions can be some of



the main concerns in these regions. In such cases, advanced learning methods, such as TL, can be ideal.

Region 5 is similar to Region 2 with additional edge nodes. Such a high density of edge nodes can enable multilayer FL with hierarchical learning methods or MARL to improve latency and user data privacy. Additionally, collaborative learning between proximal edge nodes and VUs can also be available for strugglers, allowing more users to participate in DL training.

Note that, although Figure 2.3 shows only one case of each IoV scenario, such conditions can be replicated throughout the service areas, and thus the orchestrator needs to consider the different groups of users simultaneously while assigning the slices. However, the scenarios are not limited to those stated and one may stumble upon a combination of the physical conditions illustrated in Figure 2.3. In general, we can use TL and GADMM for latency-critical tasks, SL and Hierarchical Learning for computationally intensive tasks, and FD and CFL for those that require privacy preservation. Subsequently, we can dedicate a slice to the scenario based on the conditions and applications the user demands. Note that the same physical infrastructure node (e.g., satellite) can be used for multiple logical slices at the same time.

## 2.4 Case Study

### 2.4.1 Scenario Description

The introduced DLaaS framework holds considerable potential in enhancing performance, bolstering flexibility, ensuring scalability, and fostering heightened intelligence within VNs. This study conducts a rigorous evaluation of the efficacy of

the DLaaS concept in a resource-constrained IoV scenario, depicted in Figure 2.3, within the Matlab environment. Assessment focuses on discerning performance improvements relative to conventional approaches. In this scenario, VUs are spatially distributed throughout a service area, seeking various DL slices aligned with their localized environmental conditions and application requirements. Five distinct regions are considered, each expressing a demand for a unique set of five DL slice types. As elucidated in the preceding section, the slice allocation encompasses SL for Region 1, GADMM for Region 2, FD for Region 3, TL for Region 4, and Multilayer FL for Region 5.

### 2.4.2 Function Deployment

Each requested DL slice is modeled through a chain of seven learning functions that include DAF, DPrF, DLF, DP<sub>s</sub>F, DCF, GMUF and DMIF. A multi-layered integrated T/NTN infrastructure including vehicular, RSU, UAV, HAP, and satellite layers is considered. From a functional deployment perspective, Region 5, corresponding to Slice 5, is configured with Multilayer FL, a methodology designed to execute FL across distinct layers of edge devices. Specifically, end devices use their data sets to perform the learning process. The deployment involves the placement of DAF, DPrF, and DLF on a VU layer, facilitating the learning process. Subsequently, the acquired parameters are communicated to neighboring edge nodes, represented by RSUs, via DP<sub>s</sub>F to optimize communication costs. RSUs collect data from VUs and perform aggregation operations with the assistance of DCF and GMUF. The GMUF results are then transmitted to the upper layer, where the relevant functions reside. The deployment of the function for the third slice utilizing FD mirrors the FL process described. For Slice 1, employing SL, a data parallelization technique

divides the learning process between the device and edge layers, implementing learning functions on both levels. In addition, the HAP layer is employed to formulate a global model. In the case of Slice 2, which employs GADMM, collaborative learning occurs across user devices, with appropriate functions located on the VUs cloud. The upper layers contribute to the creation of a generalized global model by aggregating local models. Slice 4, which caters to latency-critical applications, adopts a TL-based DL approach, integrating past learning experiences into current cycles to mitigate learning process costs. The learning process is distributed across various edge layers.

### 2.4.3 Network Resources and Simulation Parameters

Each layer has limited computational, storage, and communication resources available to implement the DL slices, which are indicated in Table 2.2. These values reflect the current and anticipated capabilities of NTN devices. For example, VUs are typically equipped with mobile processors that can provide up to 1 TFLOPS of performance. RSUs tend to be more powerful than VUs, boasting computational capabilities of up to 3 TFLOPS. LAPs and HAPs are typically equipped with specialized networking hardware that can provide up to 5 TFLOPS. Satellites, on the other hand, are constrained by available power and cooling, and their computational capacities typically range from 7 TFLOPS to 10 TFLOPS.

TABLE 2.2: Network resource allocation across layers.

| <b>Integrated<br/>T/NTN<br/>Layer</b> | <b>Computation<br/>Resources<br/>(TFLOPS)</b> | <b>Communication<br/>Resources<br/>(Mbps)</b> | <b>Storage<br/>Resources<br/>(GB)</b> |
|---------------------------------------|---|---|---------------------------------------|
| VU                                    | 1   | 20  | 10                                    |
| RSU                                   | 3   | 40  | 30                                    |
| LAP                                   | 3   | 30  | 10                                    |
| HAP                                   | 5   | 50  | 50                                    |
| Satellite                             | 7   | 90  | 100                                   |

The communication bandwidth values represent the current capabilities of Long-Term Evolution (LTE) and 5G and beyond cellular networks. VUs are typically connected to 5G networks with bandwidths of up to 20 Mbps. RSUs can connect to higher speed networks with bandwidths of up to 40 Mbps. LAPs and HAPs can also connect to higher-speed networks, with bandwidths of up to 50 Mbps. Satellites, however, are limited by the available bandwidth of the satellite link, and their bandwidth typically ranges from 90 Mbps to 100 Mbps.

The storage resource values represent the current capabilities of flash memory and solid-state drives. VUs typically have storage capacities of up to 10 GB. RSUs can have storage capacities of up to 30 GB. LAPs and HAPs can have storage capacities of up to 50 GB. Satellites, on the other hand, are constrained by the available storage space in the satellite, and their storage capacities typically range from 100 GB to 200 GB.

These values were employed as a starting point for our simulations and analyses. Of course, the specific values that one utilizes will depend on the specific applications

and scenarios that they are considering. Finally, the LAP, HAP, and LEO nodes are located at distances of 1.2, 20, and 1000 km from the Earth's surface, respectively. Shannon's channel capacity formula is adopted to model the channel between different layers [75].

#### 2.4.4 Key Performance Indicators

In the evaluation of our IoV scenario, we quantified user satisfaction across various KPIs—Latency, Privacy, Mobility, and Computing Capacity. The Matlab simulation environment facilitated a comprehensive analysis of the performance of the DLaaS framework under various conditions. The definitions and formulas of the KPIs we use are as follows:

- **Latency:**

- **Definition:** Latency measures the delay experienced by Vehicular Users (VUs) in obtaining responses from the DLaaS framework.
- **Simulation:** The latency satisfaction, denoted as  $S_{\text{Latency}}$ , is calculated as the percentage of users for whom the latency requirements are met:

$$S_{\text{Latency}} = \frac{\text{Number of users meeting latency requirements}}{\text{Total number of users}} \times 100\%$$

- **Privacy:**

- **Definition:** Privacy represents the level of data security and confidentiality maintained during DL processes.
- **Simulation:** Privacy satisfaction, denoted as  $S_{\text{Privacy}}$ , is calculated similarly based on the percentage of users for whom privacy requirements are met.

$$S_{\text{Privacy}} = \frac{\text{Number of users meeting privacy requirements}}{\text{Total number of users}} \times 100\%$$

- **Mobility:**

- **Definition:** Mobility reflects the ability of VUs to maintain seamless connectivity while traversing diverse geographical regions.
- **Simulation:** Mobility satisfaction, denoted as  $S_{\text{Mobility}}$ , is calculated based on the percentage of users meeting mobility requirements.

$$S_{\text{Mobility}} = \frac{\text{Number of users meeting mobility requirements}}{\text{Total number of users}} \times 100\%$$

- **Computing Capacity:**

- **Definition:** Computing capacity denotes the ability of the DLaaS framework to handle computational demands efficiently.
- **Simulation:** Computing capacity satisfaction, denoted as  $S_{\text{Computing}}$ , is calculated similarly based on the percentage of users meeting computing capacity requirements.

$$S_{\text{Computing}} = \frac{\text{Number of users meeting computing capacity requirements}}{\text{Total number of users}} \times 100\%$$

### 2.4.5 Impact of Multiple Slices on User Satisfaction:

In Figure 2.4, user satisfaction with respect to Latency, Privacy, Mobility, and Computing Capacity is illustrated. Different types of DL slices were used, each tailored to specific computational and communication characteristics, as detailed in Table 2.2. The results demonstrate that with five slices, all considered KPIs achieve high

satisfaction levels for all users. Conversely, with only one slice, satisfaction levels drop significantly, underscoring the effectiveness of the proposed multi-slice DLaaS framework.

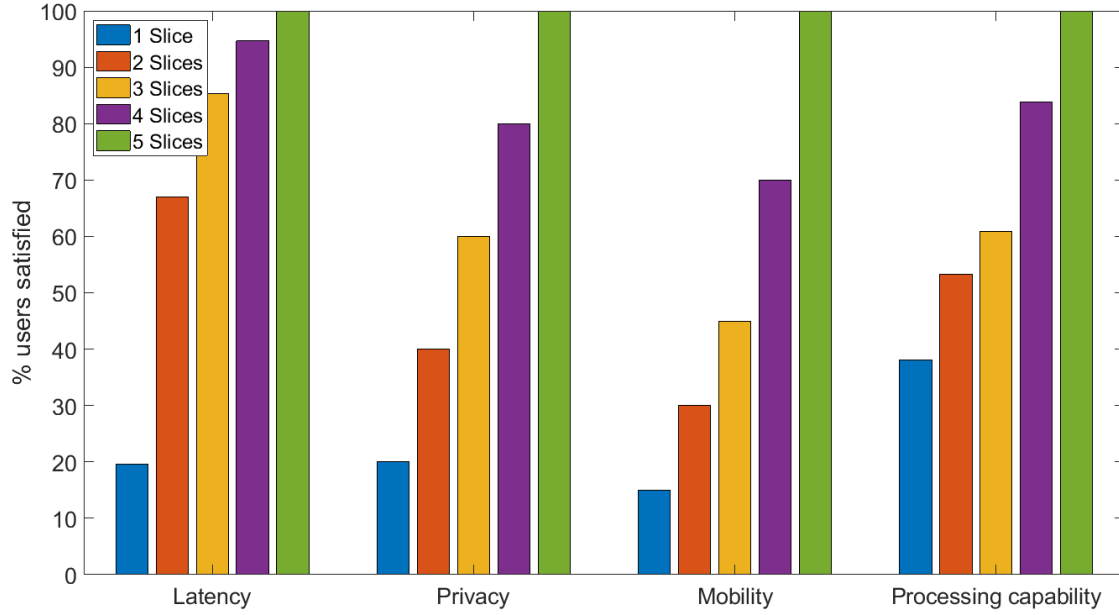


FIGURE 2.4: User satisfaction in different KPIs vs the number of slices.

#### 2.4.5.1 Mobility

In Figure 2.5, we observe the impact of varying the number of slices on user satisfaction with respect to mobility for different numbers of VUs. The results highlight a notable trend. As the number of slices increases, there is an evident improvement in mobility satisfaction. This improvement is attributed to the ability to cover larger areas of the ground efficiently. With multiple slices, the service infrastructure can span wider geographical regions, allowing VUs to traverse expansive areas without experiencing loss of connectivity to the server or higher-layer entities, such as UAVs.

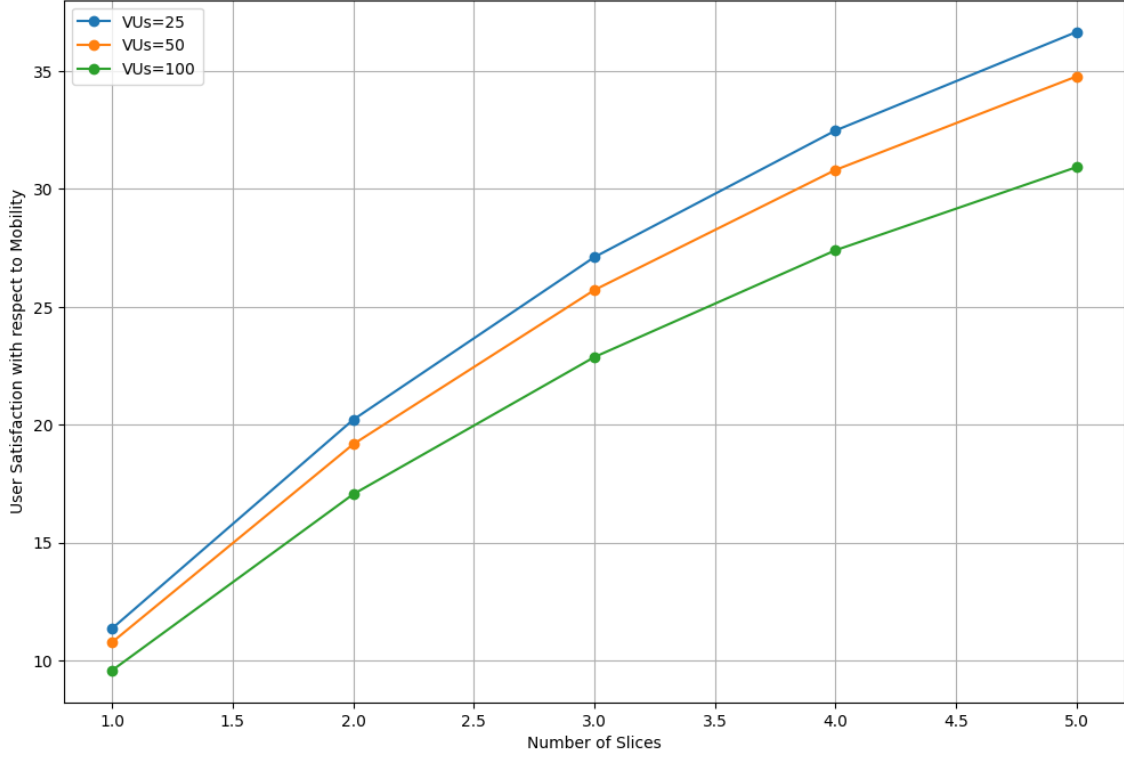


FIGURE 2.5: User satisfaction with respect to mobility versus the number of slices for different number of VUs.

In particular, when only one slice is utilized, the coverage of the entire area is based on a single slice and the FL method. This limitation presents challenges in efficiently serving diverse VU mobility patterns. However, as the number of slices increases, the system gains the ability to take advantage of different layers of NTN, leading to a significant increase in user satisfaction. This is particularly advantageous as the number of VUs increases, demonstrating the adaptability of the proposed framework to scale and accommodate growing demands without compromising user satisfaction in terms of mobility.

#### 2.4.5.2 Processing capacity

The insights derived from Figure 2.6 accentuate the positive impact of employing



multiple slices on user satisfaction with the average processing capacity, especially in the presence of varying numbers of VUs. The figure manifests a clear trend in which the use of the slices contributes to a substantial improvement in the overall satisfaction of the VUs with respect to the processing capacity.

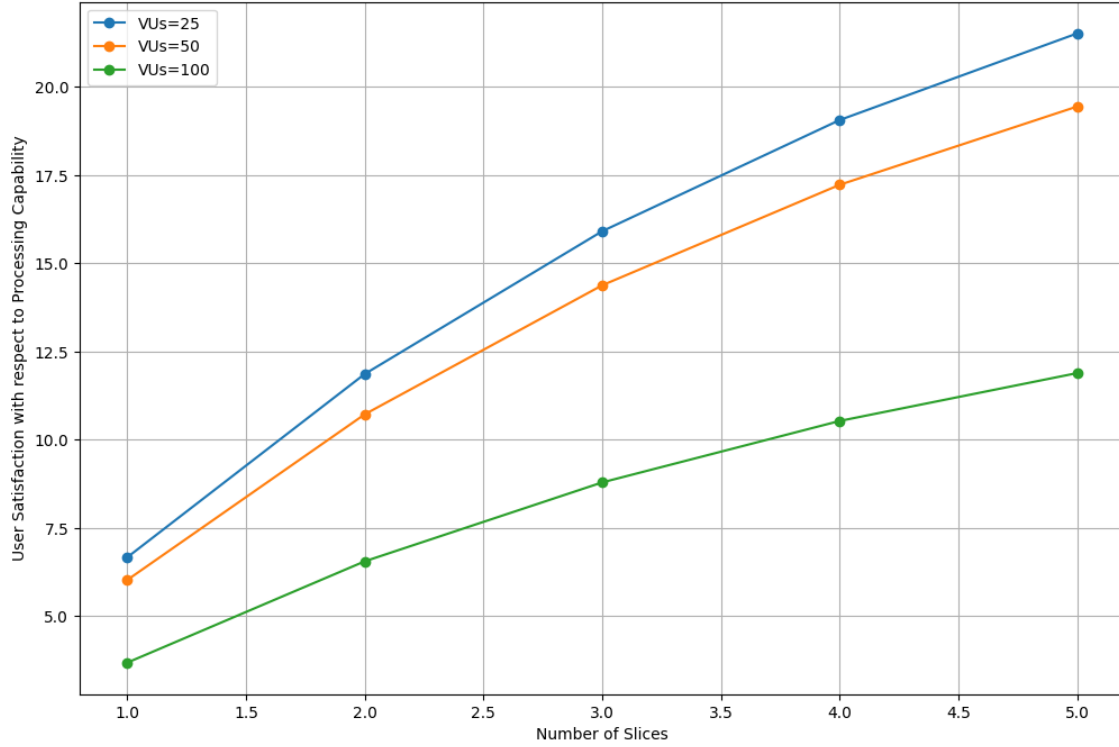


FIGURE 2.6: User satisfaction with respect to average processing capability versus the number of slices for different number of VUs.

As expected, the provision of more resources through the implementation of slices in each layer empowers the system to quickly meet the increasing demands of a growing VU population. This resilience is particularly noteworthy, as the figure demonstrates sustained improvements in processing capability satisfaction even with a higher influx of VUs. The results reiterate the superiority of the proposed DLaaS structure in efficiently catering to the computational needs of a dynamic and expanding user base.

### 2.4.5.3 Average Latency

Figure 2.7 delves into the realm of latency satisfaction, shedding light on the influence of the number of slices on the user experience, depending on the varying number of VUs. The results portray enhanced performance in terms of latency satisfaction as the number of slices increases. This improvement can be attributed to the increased bandwidth available for communication, resulting in reduced communication and transmission delays. Additionally, the increased resources in each slice and layer contribute to serving VU demands with reduced computation delay.

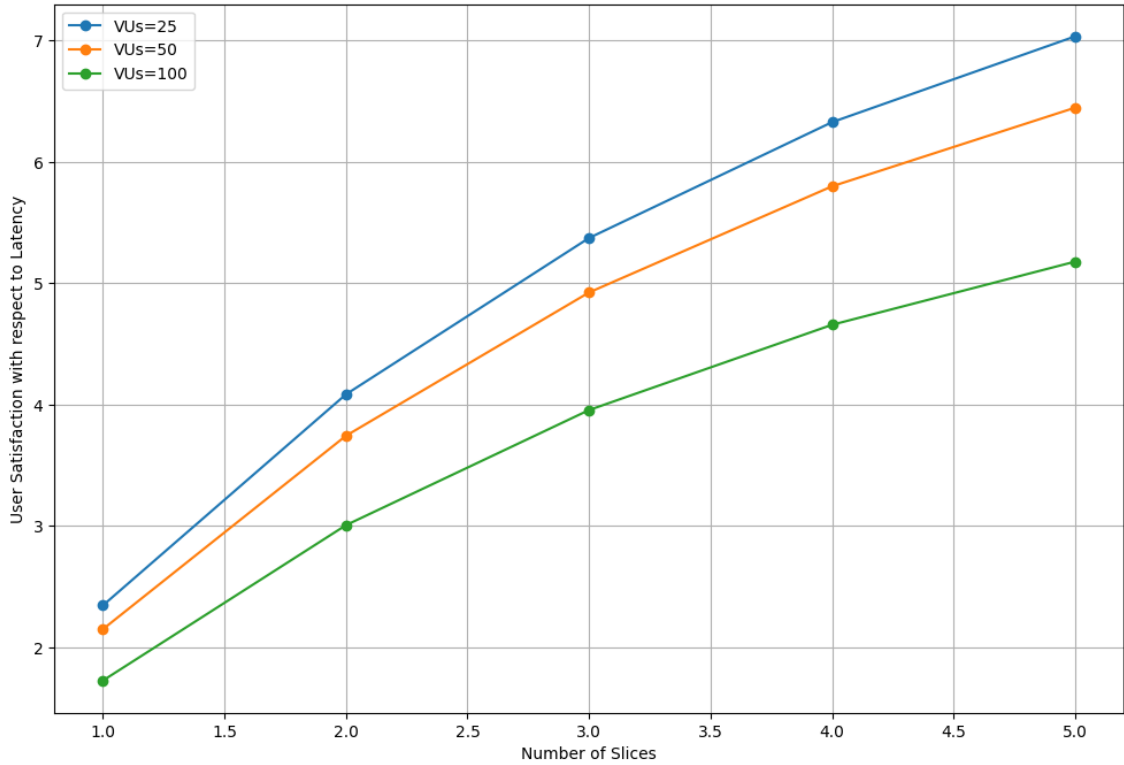


FIGURE 2.7: User satisfaction with respect to average latency versus the number of slices for different number of VUs.

The observed decrease in total latency, resulting from optimized communication and computation resources, underscores the efficacy of the proposed DLaaS structure. Interestingly, the figure reveals that even with a doubling of the number of VUs,

the latency KPI does not undergo a proportional decrease. This phenomenon signifies the strategic utilization of additional resources made available through multiple slices, showcasing the system's resilience to scalability challenges.

### 2.4.6 Average Response Time

On the other hand, we would like to show how slicing can be beneficial in reducing the time required to respond to the varying tasks demanded by users. To achieve this, we assume that the Probability Density Function (PDF) of the requested tasks is Poisson distributed with a parameter  $\lambda$ , which represents the average frequency of requests for each VU. The expected time value required for the slices to reconfigure to a new state for the requested services is reported, showing how much time is needed between consecutive requests to deploy a slice. Figure 2.8 shows this variable versus the number of slices for five different values of  $\lambda$  in requests per second. This figure leads us to conclude that the more slices, the lower the reconfiguration frequency, and five slices are enough for the reconfigurability to go to zero.

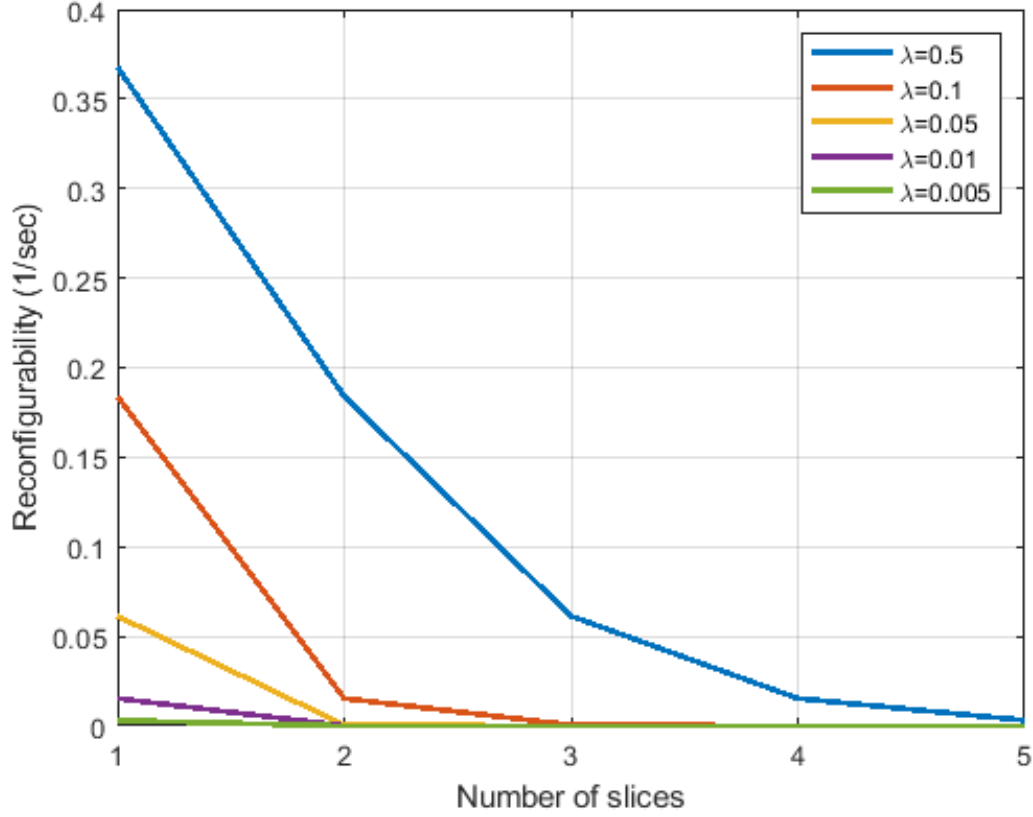


FIGURE 2.8: Reconfigurability vs. the number of slices.

## 2.5 Conclusion

In this work, we have proposed a DLaaS concept to implement various DL slices in distributed vehicular environments through virtualization on NTN. We have analyzed various technologies, including EC, distributed computing/communication, NS, and different DL methods. An end-to-end functional decomposition of typical DL methods and their possible implementation as slices over the distributed networking platforms is discussed. A detailed case study of a typical vehicular scenario is considered. Through simulation, it has been shown that the proposed DLaaS concept can have several advantages in terms of user satisfaction, flexibility, scalability,

---

performance boosts, and added intelligence. Some key challenges of the proposed DLaaS concept are also provided to motivate future research.

## Chapter 3

# Multi-Layer Distributed Learning for 6G ITS

Some of the content of this chapter is based on the following articles [76, 77, 78, 79, 80]:

1) **David Naseh**, Arash Bozorgchenani, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Unified Distributed Machine Learning for 6G Intelligent Transportation Systems: A Hierarchical Approach for Terrestrial and Non-Terrestrial Networks." *Network* 5, no. 3 (2025): 41.

2) **David Naseh**, Swapnil Sadashiv Shinde, Daniele Tarchi, and Tomaso DeCola. "Distributed Intelligent Framework for Remote Area Observation on Multilayer Non-Terrestrial Networks." In *2024 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pp. 1-6. IEEE, 2024.

3) **David Naseh**, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Multi-layer distributed learning for intelligent transportation systems in 6G aerial-ground integrated

networks.” In *2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pp. 711-716. IEEE, 2024.

4) **David Naseh**, Swapnil Sadashiv Shinde, and Daniele Tarchi. ”Distributed learning framework for earth observation on multilayer non-terrestrial networks.” In *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, pp. 1-2. IEEE, 2024.

5) **David Naseh**, Swapnil Sadashiv Shinde, and Daniele Tarchi. ”Enabling intelligent vehicular networks through distributed learning in the non-terrestrial networks 6G vision.” In *European Wireless 2023; 28th European Wireless Conference*, pp. 136-141. VDE, 2023.

### 3.1 Introduction

With the 6G vision set to shape society in the 2030s into a more advanced, digitized, fully connected, and intelligent world, transportation networks are undergoing a significant transformation, converging into Intelligent Transportation Systems (ITS) [81]. The field of ITS is rapidly expanding, with the aim of improving the safety, efficiency and sustainability of traditional transportation systems by using technologies such as Machine Learning (ML), the Internet of Things (IoT), and advanced communication modes [82]. The presence of IoT subsystems facilitates the generation of large amounts of data from various components of the ITS infrastructure over time [83]. These data can be used to provide intelligent solutions, optimize traffic management, improve user experiences, and improve environmental outcomes. However, significant challenges arise in the acquisition and analysis of data from various sources, including vehicles, sensors, and traffic cameras [84].

Recent advancements in Distributed Learning (DL) techniques offer promising solutions to these challenges. Among them, Federated Learning (FL) has emerged as a particularly effective paradigm for training ML models in a distributed manner, enabling collaboration while maintaining data privacy [75]. FL facilitates shared model training without requiring the exchange of individual data among multiple devices. Each device trains a local model on its data and then updates a central server with its progress. The server aggregates the updates and applies them to a global shared model. However, one of the main drawbacks of FL is that each client must train the entire ML model, which can be resource-intensive and impractical for clients with limited resources, particularly in ITS environments, especially when employing complex models like DNNs [85]. Furthermore, the iterative transmission of local and global model parameters has raised new privacy concerns, including risks of data poisoning and model inversions, highlighting the need for improved security mechanisms in FL frameworks [86].

To mitigate the limitations of FL, Split Learning (SL) presents an alternative approach. SL allows complex ML models to be trained by dividing them into two parts, with each part trained on a client or a server using local data from distributed clients [87]. This method significantly reduces the computational burden on resource-limited devices, as only a portion of the model is trained locally and communication is limited to the activation of the cut layer. Consequently, SL enhances model privacy by preventing direct access to the entire model on either side. Recent research has demonstrated the potential of SL-inspired frameworks to enhance the efficiency and scalability of FL approaches, allowing the training of more sophisticated DNNs while maintaining privacy and reducing costs [88].

In addition to these methods, Transfer Learning (TL) from the meta-learning family has gained attention for its ability to improve training efficiency by facilitating



Knowledge Transfer (KT) from previous tasks to new related tasks [89]. TL can accelerate convergence rates, reduce reliance on labeled data, and enhance the robustness of ML models in various vehicular scenarios. The integration of TL with FL, particularly in resource-constrained ITS environments, presents a valuable opportunity to leverage previous knowledge and improve model performance [90]. However, existing works have yet to comprehensively address how these techniques can be optimally combined to maximize their benefits in dynamic and heterogeneous environments.

Within this framework, Earth Observation (EO) has emerged as a complementary technology that can significantly enhance the capabilities of ITS. EO involves the collection and analysis of satellite and aerial data, providing critical information on traffic conditions, weather patterns, and environmental changes [91]. This integration allows ITS to benefit from real-time data streams, facilitating improved traffic management, hazard detection, and overall situational awareness [92]. However, integrating EO with ITS presents its own set of challenges, including the need for efficient data fusion techniques from heterogeneous sources, real-time processing capabilities, and advanced ML methodologies to analyze vast amounts of EO data effectively [93].

In the context of the 6G vision, Non-Terrestrial Networks (NTNs) play a pivotal role in providing global coverage and capacity enhancements for traditional TNs. Various NTN platforms, including High-Altitude Platforms (HAPs), offer significant advantages for vehicular users (VUs), such as reduced transmission distances, improved coverage, and flexible deployment options [94]. These characteristics make NTNs particularly well-suited for supporting intelligent solutions in ITS by facilitating efficient DL methods and enhancing connectivity in remote or underserved areas [95].

In the realm of ITS-oriented DL, several hybrid frameworks have been explored, each contributing valuable insights while leaving important gaps. For example, ref. [96] introduces a hybrid of FL and SL tailored to ITS, enabling personalized model training across vehicles while preserving data privacy; however, it lacks TL integration and does not consider the latency and resource constraints of multilayer network deployments. Another relevant study, ref. [97] evaluates the security and effectiveness of FL, SL, and TL in vehicular networks; while it clarifies which techniques are feasible for privacy preservation, it remains a feasibility study and does not propose a unified framework that combines these paradigms with scalability in multilayer NTN settings. A third work [98] develops a variant of Federated Split Learning (FSL) that addresses the challenge of vehicles lacking labeled data by splitting the model between onboard units and RSUs, preserving both data and label privacy; however, it does not address dynamic grouping, hierarchical aggregation, or real-time adaptation in 6G NTN-enhanced ITS.

### 3.1.1 Gaps in the Literature and Motivations

Table 3.1 summarizes representative prior works and highlights the main differences relative to our approach. Although existing studies have advanced FL, SL, TL and hybrid schemes for edge and vehicular settings, several important gaps remain that limit their applicability to multilayer 6G ITS and NTN-enabled Earth Observation. The most salient gaps are as follows:

1. **Under-utilization of multilayer NTN architectures:** Many prior frameworks assume single-tier or purely terrestrial deployments and therefore do not exploit the potential benefits of coordinated processing across RSUs, HAPs,

TABLE 3.1: Comparison of the most related works and gaps/differences relative to our work.

| Ref. | Contributions  | Differences With Our Work  |
|------|--|--|
| [85] | Multitask FL with hybrid client selection/aggregation in edge networks.                      | Pure FL (no SL/TL); no multilayer hierarchy or NTN integration; not tailored to ITS.   |
| [87] | Evaluates and optimizes DL techniques for IoT; benchmarking focus.                           | Not ITS-centric; not hierarchical FL/SL over T/NTNs nor real-time ITS latency study.   |
| [88] | Wireless Distributed Learning: hybrid split+federated approach.                              | No T/NTN tiering, no EO/ITS integration, and no multilayer ITS latency/accuracy study. |
| [90] | Federated Transfer Learning for cross-domain sensing with resource efficiency.               | No ITS; no SL or multilayer NTN; narrower modality than our EO/vehicular perception.   |
| [94] | 6G-enabled advanced transportation systems; network capabilities and use cases.              | No DL pipeline or a multilayer T/NTN latency/accuracy quantification.                  |
| [95] | Explores the potential of NTNs in next-gen ITS; positioning paper.                           | Lacks concrete DL training/aggregation across layers and empirical ITS evaluation.     |
| [96] | Personalized FSL for ITS-distributed training: single-layer training.                        | No hierarchy, no EO integration, and no unified FL/SL/TL orchestration across T/NTNs.  |
| [97] | Feasibility of SL/TL/FL for security in ITS; SL outperforms FL/TL baselines; security focus. | No ITS; no hierarchical T/NTN training or EO-ITS fusion; no multi-tier aggregation.    |
| [98] | FSL with data/label privacy in vehicular networks.   | No multilayer architecture and no joint FL+SL+TL orchestration.                        |

and satellites. This omission reduces opportunities for latency-aware placement, load balancing across tiers, and resilience in coverage-challenged regions, all critical for large-scale ITS.

2. **Limited onboard and in-space data processing:** Existing works rarely address efficient processing of EO data within non-terrestrial platforms or the trade-offs introduced by performing model tasks in space (e.g., at HAPs/LEO

nodes). Without concrete strategies for in-space inference/aggregation, systems face increased communication cost or delayed decision-making for time-sensitive ITS applications.

3. **Insufficient integration of DL paradigms for resource-constrained heterogeneous clients:** While FL, SL, and TL have been studied individually (and some combinations have been proposed), there is a lack of comprehensive frameworks that jointly leverage model partitioning, transfer initialization, and hierarchical aggregation to simultaneously address privacy, compute constraints, and converge quickly in highly heterogeneous ITS fleets.

The emerging 6G paradigm, characterized by ubiquitous and seamless T/NT connectivity, extreme low-latency links, pervasive edge intelligence, and AI-native network management, directly addresses the limitations noted above. Native NTN integration in 6G enables hierarchical, multilayer orchestration (e.g., RSU/UAV/HAP/satellite) that supports latency-aware placement and multi-tier aggregation. Likewise, 6G's stronger edge computing and deterministic service capabilities make practical the notion of in-space or at-edge model processing for EO and real-time ITS tasks, reducing the need for bulky raw-data transfers. Finally, the AI-native control plane and advanced resource-slicing features foreseen for 6G facilitate dynamic model partitioning and adaptive use of FL/SL/TL mechanisms across heterogeneous clients. These 6G capabilities therefore motivate and make feasible the design objectives of this chapter: a multilayer joint T/NT, latency-aware DL framework that unifies Federated, Split, and Transfer Learning and maps model components to the most appropriate network tier, improving scalability, privacy, and real-time performance for ITS and NTN-based EO scenarios.

To address these gaps, we propose two comprehensive methodologies: Federated Split Transfer Learning (FSTL) and its generalized version, Generalized Federated

Split Transfer Learning (GFSTL). Both methods are designed for deployment in T/NTNs (or joint air-ground networks) and will be applied to three specific applications: vehicular scenarios, EO and then their combination as an ITS scenario. The FSTL framework integrates the advantages of FL, SL, and TL, providing a scalable solution for training ML models in resource-constrained environments [80]. Meanwhile, GFSTL introduces a flexible approach that allows the independent use of FL and SL servers, catering to diverse user requirements. This novel architecture not only reduces latency and enhances the number of participants in the federated training process, but also improves overall model accuracy and privacy protection. We will evaluate the effectiveness of the proposed methodologies through simulations in typical 6G ITS scenarios, using advanced architectures like ResNet and You Only Look Once (YOLO) [99], demonstrating their superiority over traditional FL methods in terms of convergence rates, training accuracy, and overall latency.

### 3.1.2 Key Contributions

This chapter introduces GFSTL and FSTL, two novel DL frameworks designed for multilayer 6G ITS and NTN-enabled EO. The main contributions are summarized as follows:

1. **New hybrid learning frameworks (FSTL and GFSTL):** We propose FSTL and its generalized version (GFSTL). FSTL integrates FL, SL, and TL into a single pipeline tailored for resource-constrained edge nodes. GFSTL extends FSTL by supporting multiple independent FL/SL server placements and hierarchical aggregation across Road-Side Units (RSUs) and HAPs, enabling flexible deployment across Terrestrial and Non-Terrestrial Network layers (see Sections 3.3 and 3.4).

2. **Exploitation of multilayer T/NTN architectures for scalable DL:** We explicitly design the frameworks to exploit the multilayer structure of NTN (RSUs, UAVs, HAPs, Low/Medium Earth Orbit (LEO/MEO) satellites) so that computation and aggregation are performed at the most appropriate network tier (see the multilayer architectures in Section 3.7). This hierarchical design reduces wall-clock training time, improves scalability, and increases the number of feasible participants in federated training (see Section 4.2.6 and the latency analysis in Section 3.6).
3. **Efficient Data Processing for Real-Time EO/ITS Applications:** The work proposes novel strategies for real-time data processing in space to optimize the utilization of EO/ITS data. By leveraging GFSTL's flexible architecture, we address the inefficiencies in current EO/ITS systems, enabling more effective data transmission and model updates, even in highly dynamic environments with limited resources (Section 3.7).
4. **Application to Diverse Use Cases and Comprehensive Evaluation:** The proposed methodologies, system architecture, training process, added benefits, and challenges are introduced and evaluated across three distinct use cases: vehicular scenarios (Section 3.7.1), EO (Section 3.7.2), and their integration into a unified ITS scenario (Section 3.8). This work also provides a comprehensive latency analysis for a multilayer 6G ITS environment (Section 3.6), showcasing the practical benefits of our approach in large-scale, real-world deployments.
5. **Enhanced Latency, Accuracy, and Privacy in DL:** Our proposed methods significantly reduce latency and enhance training accuracy compared to traditional FL-based frameworks. The introduction of flexible server configurations in GFSTL further improves the number of participants in the FL

process while ensuring robust privacy protection. These advances are demonstrated through simulations in typical 6G ITS scenarios, where the proposed methodologies outperform traditional FL techniques in terms of convergence rates, model accuracy (Sections 3.9.2 and 3.9.3), and overall system latency (Section 3.9.4).

#### 6. **Practical split/transfer parameterization for ITS and EO tasks:**

We provide a concrete parameterization (model cut-points, smashed-representation sizing, and YOLO/ResNet choices) that maps model components to node capabilities (RSU/HAP/edge) (Section 3.9.1), balancing accuracy, computation, and communication. This parameterization is validated empirically in Section 4.2.6 and supports real-time ITS perception tasks using compact intermediate representations.

#### 7. **Comprehensive performance and latency study under unified experimental settings:**

We evaluate FL, SL, FSL, FSTL, and GFSTL on a representative ITS scenario using the same hardware/network assumptions to ensure a fair comparison. This study quantifies trade-offs in accuracy, convergence speed, per-round latency, and communication volume and demonstrates our proposed frameworks' improvements in these dimensions (results and discussion in Sections 4.2.6 and 3.10).

The above contributions jointly advance the state of the art by presenting scalable, privacy-preserving, and latency-aware DL solutions that are directly applicable to multilayer 6G ITSs and NTN-based EO systems.

### 3.1.3 Organization of the Chapter

This chapter is organized as follows: Section 3.2 provides an overview of existing DL frameworks, including FL, SL, FSL, and TL, discussing their advantages and limitations. Section 3.3 details the proposed FSTL framework, outlining its architecture and training process and showcasing its application in ITSs. Building upon this, Section 3.4 introduces the GFSTL framework, presenting its architecture and training algorithm for scalable, multi-group network environments. Section 3.7 demonstrates the versatility of GFSTL through two distinct use cases: a vehicular Aerial–Ground Integrated Network (AGIN) and an NTN-based EO system, followed by a comprehensive latency analysis for a unified multi-layer 6G ITS scenario. Section 4.2.6 presents the simulation setup and parameters, evaluation metrics, and the results validating the proposed methodologies. Finally, Section 3.10 provides a discussion of the findings, limitations, and future directions, with Section 3.11 concluding the chapter.

## 3.2 Distributed Learning Frameworks

DL has become an essential approach for enabling collaborative model training across multiple devices without requiring centralized data storage. This method is particularly valuable in scenarios like ITS, NTNs, and EO, where large volumes of data are distributed among many devices, and privacy concerns or bandwidth limitations make data centralization impractical. By allowing each device to process its data locally and share only model updates with a central server, DL frameworks ensure that privacy is maintained while enabling efficient, large-scale model training. Some of the most widely adopted DL techniques include FL, SL, FSL, and FSTL, each



offering unique benefits to address the challenges of distributed data environments [17].

### 3.2.1 Federated Learning

FL is one of the foundational methods for DL, allowing multiple clients (e.g., VUs in an ITS) to collaboratively train a shared ML model without sharing their raw data. FL ensures privacy and reduces the bandwidth needed for communication, making it well-suited for environments where data privacy is critical, such as ITS [44]. Each client performs model training on its local data and only transmits model updates (such as gradients) to a central server for aggregation.

Consider a system with  $N$  distributed users defined through the set  $\mathcal{U} = \{u_1, \dots, u_N\}$ , where each user  $i$  has a local dataset  $\mathcal{D}_i = \{(\mathbf{x}_k, y_k)\}$  containing  $K_i$  labeled data samples. The feature vector  $\mathbf{x}_k \in \mathbb{R}^n$  is the  $k$ -th sample, while  $y_k$  is its corresponding label. The task is to train a global model  $W^p$  by minimizing a global loss function  $L^p$  across all clients without sharing the raw data.

During each round of communication, users update their local models by minimizing their local loss functions  $L_i^p(\mathcal{D}_i, W_t^p)$ , where  $W_t^p$  represents the global model in round  $t$ . The updated local model parameters of each user are sent to the server, which aggregates them using a weighted averaging scheme, such as FedAvg [100]:

$$W_{t+1} = \frac{1}{N} \sum_{i=1}^N W_{i,t}^p \quad (3.1)$$

The process continues until the model converges or a stopping criterion is met, such as a target loss value. The advantages of FL are as follows:

- **Data Privacy:** FL allows users to keep their raw data local, making it ideal for privacy-sensitive scenarios.
- **Reduced Communication Overhead:** Only the model parameters are exchanged, not the raw data, saving bandwidth.
- **Adaptability:** FL can continuously improve models with real-time data from users, making it responsive to dynamic environments.

The disadvantages are the following:

- **Communication Latency:** Multiple rounds of communication is required, increasing the overall latency, especially in bandwidth-constrained networks.
- **Convergence Speed:** Achieving convergence may require many iterations, resulting in high computational and communication costs for resource-constrained users.

As FL addresses privacy and communication challenges, its iterative nature introduces certain limitations, particularly in environments where latency or computation resources are limited [101]. To overcome these challenges, more advanced frameworks like SL have been introduced. The interested reader could refer to [102] for a comprehensive analysis of FL algorithms.

### 3.2.2 Split Learning

Although FL provides a solid foundation for DL, it can place significant computational and communication burdens on clients, especially when training deep models [103]. To mitigate these issues, SL divides the learning process between clients and

servers. In SL, each client processes only the lower layers of the model, while the server handles the remaining computations. This reduces the computational load on clients and enhances privacy by keeping raw data within the client device [104].

In the context of an ITS scenario, consider a model  $W^P$  that is divided into two components: (i) the client-side model  $S(\cdot)$ , which each user or client maintains locally, and (ii) the server-side model  $M(\cdot)$ , responsible for completing the forward and backward passes on the server. This split architecture allows resource-constrained devices, such as VUs, to perform only the initial stages of the computation, reducing their computational and storage requirements.

The SL process involves the following steps for a single forward-backward round:

1. Each user processes its local data  $\mathcal{D}_i$  using the client-side model  $S(\cdot)$ , which generates intermediate activations:

$$H_i = S(\mathcal{D}_i) \quad (3.2)$$

These intermediate activations  $H_i$  represent compressed feature representations and are transmitted to the server.

2. Upon receiving  $H_i$ , the server performs the forward pass using its merge model  $M(\cdot)$  to produce predictions  $\hat{y}_i$ :

$$\hat{y}_i = M(H_i) \quad (3.3)$$

This completes the forward pass for the entire model  $W^P$  using both client- and server-side components.

3. Next, the server begins the backward pass on the server-side model  $M(\cdot)$  to compute the gradients of its model parameters based on the loss function,

typically using a method such as Stochastic Gradient Descent (SGD). The server then calculates the gradient of the intermediate activations  $H_i$ , denoted as  $\nabla H_i$ , and transmits this gradient back to the user.

4. Finally, the user completes the backward pass locally on its split model  $S(\cdot)$  using  $\nabla H_i$ . This enables each client to update their local model parameters for  $S(\cdot)$  based on its unique data, thus maintaining data privacy and security.

This split process enables each user to perform forward and backward propagation collaboratively with the server, forming a full round of training over the model  $W^P$ . By structuring the model in this manner, SL allows resource-constrained client devices to participate in training without handling the full computational load. The server's ability to process intermediate activations reduces client-server communication overhead while preserving data privacy by ensuring that raw data remains local to each client.

The advantages of SL are as follows:

- **Reduced Client-Side Load:** Clients only need to compute part of the model, reducing their computational overhead.
- **Privacy Preservation:** Only smashed data (intermediate activations) is transmitted, protecting the privacy of raw data.
- **Lower Communication Cost:** Fewer data need to be exchanged between clients and the server, as only intermediate activations are shared.

The disadvantages are the following:

- **Sequential Processing:** The split model introduces latency because clients and the server must alternate between steps.

- **Reduced Model Expressiveness:** Splitting the model might reduce its capacity, potentially leading to performance degradation.

While SL reduces computational load and enhances privacy, it can still suffer from latency issues due to its sequential nature [105]. The interested reader could refer to [106] for a comprehensive analysis of SL algorithms.

### 3.2.3 Federated Split Learning

To overcome earlier limitations, a novel framework named FSL merges the benefits of FL and SL. FSL integrates the privacy-preserving nature of FL with the computational efficiency of SL. FSL enables clients to process part of the model locally (as in SL) and then uses FL's Federated Averaging (FedAvg) method to aggregate the intermediate representations across clients. This hybrid approach leverages the advantages of both frameworks while addressing their shortcomings [107].

In FSL, each user computes intermediate activations  $H_i = S(\mathcal{D}_i)$  locally, which are sent to the server for aggregation. The server merges these activations using a FedAvg scheme:

$$H_{merged} = \frac{1}{N} \sum_{i=1}^N H_i \quad (3.4)$$

The merged activations are then used to update the global model, and the updated parameters are distributed back to the users. By combining FL's distributed training process with SL's efficient model-splitting approach, FSL can achieve better scalability and improved privacy. The advantages of FSL are as follows:

- **Hybrid Approach:** FSL benefits from both FL and SL, enabling DL with lower communication costs and enhanced privacy.

- **Scalability:** The FedAvg mechanism allows the framework to scale effectively across many clients without increasing the communication burden.
- **Reduced Client-Side Load:** Like SL, FSL reduces the computational burden on clients by only requiring them to compute the lower layers of the model.

The disadvantages are the following:

- **Slow Convergence:** The nature of SL can result in slower convergence, particularly in large-scale networks.
- **High Local Training Requirements:** Clients still need sufficient local computational resources to handle their portion of the model, which may not always be feasible in resource-constrained environments.

Although FSL improves the benefits offered by both FL and SL, its implementation can suffer from delayed convergence rates, particularly in extensive deployment scenarios [108]. To further optimize the efficiency of the learning procedure, TL can be integrated with FSL, giving rise to the FSTL methodology proposed later. Before analyzing FSTL, we introduce TL, focusing on its advantages.

### 3.2.4 Transfer Learning

TL is a powerful technique that improves model performance by leveraging knowledge gained from previous tasks, enabling efficient training on new but related tasks. TL is especially useful in scenarios with limited data or heterogeneous data distributions among clients, as it reduces the training required for convergence and enhances

overall model accuracy [109]. By initializing clients with pre-trained models developed on large datasets, TL allows each client to start training from a more informed state, rather than from random initialization [110].

Formally, let  $\mathcal{D}_{source} = \{(x_i, y_i)\}_{i=1}^{N_s}$  represent the source domain with  $N_s$  samples and associated labels  $y_i$ . The objective is to learn a model  $W^{p'}$  that minimizes the loss function  $L^{source}$  on this domain:

$$L^{source} = \frac{1}{N_s} \sum_{i=1}^{N_s} \mathcal{L}(y_i, f(x_i; W^{p'})) \quad (3.5)$$

where  $\mathcal{L}$  is a chosen loss function (e.g., cross-entropy) and  $f(x_i; W^{p'})$  is the model's prediction for input  $x_i$ .

Once trained on the source domain, the model is fine-tuned on the target domain  $\mathcal{D}_{target} = \{(x_j, y_j)\}_{j=1}^{N_t}$  with  $N_t$  samples to minimize the loss function  $L^{target}$ :

$$L^{target} = \frac{1}{N_t} \sum_{j=1}^{N_t} \mathcal{L}(y_j, f(x_j; W^{p'})) \quad (3.6)$$

Within the context of FSL, TL provides substantial benefits. By assigning a pre-trained model  $W^{p'}$  to each client, the learning process is accelerated and the generalization of the model in varying datasets of clients is improved. This combination is particularly advantageous when client data distributions differ, as the initial pre-trained features and patterns offer a robust starting point for adaptation.

TL offers the following advantages:

- **Accelerated Training:** TL reduces the training time required for model convergence, which is particularly advantageous in real-time or resource-constrained applications.

- **Improved Model Accuracy:** Leveraging learned knowledge from a related domain improves model performance, especially when training data is limited or heterogeneous.
- **Enhanced Generalization:** Pre-trained models offer better adaptability across diverse client datasets, as they can transfer features to new tasks effectively.

In the transition from FSL to FSTL, TL further enhances the learning framework. Integrating TL with FSTL maximizes efficiency and scalability while preserving the data privacy and computational benefits of FSL. This synergy allows for faster and more adaptable model training in distributed environments, which is particularly beneficial for diverse applications with unique data characteristics.

### 3.3 Federated Split Transfer Learning

In the realm of FSL, TL offers significant advantages. The allocation of a pre-trained model  $W^{P'}$  to each client expedites the learning process and enhances the model's capacity to generalize across the diverse datasets of clients. This synergistic approach is especially beneficial when there are disparities in client data distributions, as the pre-trained features and patterns provide a solid foundation for adaptation. As FSL progresses to FSTL, TL further strengthens the learning architecture. The fusion of TL with FSL optimizes both efficiency and scalability while maintaining the privacy and computational benefits inherent to FSL. This collaboration facilitates quicker and more adaptable model training in distributed settings, which is particularly advantageous for a range of applications that feature distinct data properties.



In FSTL, the goal is to enhance the capabilities of DL by combining the benefits of FL, SL, and TL. The FSTL framework is particularly advantageous in scenarios where resource-constrained devices, such as VUs or UAVs, participate in collaborative learning while maintaining data privacy and minimizing communication overhead.

### 3.3.1 FSTL Architecture

The architecture of FSTL begins with a pre-trained Neural Network (NN) model  $W^{p'}$ , which consists of  $L$  layers. This model is divided at a specific cut layer  $k$  (where  $1 < k < L$ ) into two parts:

1.  $W_U^{p'}$ , the portion up to layer  $k$ , which is deployed on each user for local processing, and
2.  $W_S^{p'}$ , the remaining layers, which are managed by the server in the SL paradigm.

During training, each  $i$ -th VU processes its local dataset  $\mathcal{D}_i$  using the client-side split model  $W_U^{p'}$ , generating intermediate representations  $H_i$ . These representations are then transmitted to the SL server-side model  $W_S^{p'}$ , which processes the intermediate data and updates the global model parameters. The split structure allows users to handle computationally lighter tasks while offloading the most demanding parts to the server, thereby reducing local resource consumption.

By introducing TL into the FSL paradigm, the FSTL framework leverages the advantages of pre-trained models on the client side. The pre-trained layers up to the cut layer  $k$  capture generalizable features and patterns that are useful across different tasks. Meanwhile, the remaining layers on the server allow for further refinement

and collaborative learning among distributed users. As a result, FSTL benefits from more efficient KT, improved model performance, and faster convergence compared to traditional SL or FL methods.

A key improvement in the FSTL framework over traditional SL is its capacity to enable parallel processing by allowing all clients to communicate simultaneously with both the SL and FL servers. This architecture, as illustrated in Figure 3.1, offers several distinct advantages over sequential methods:

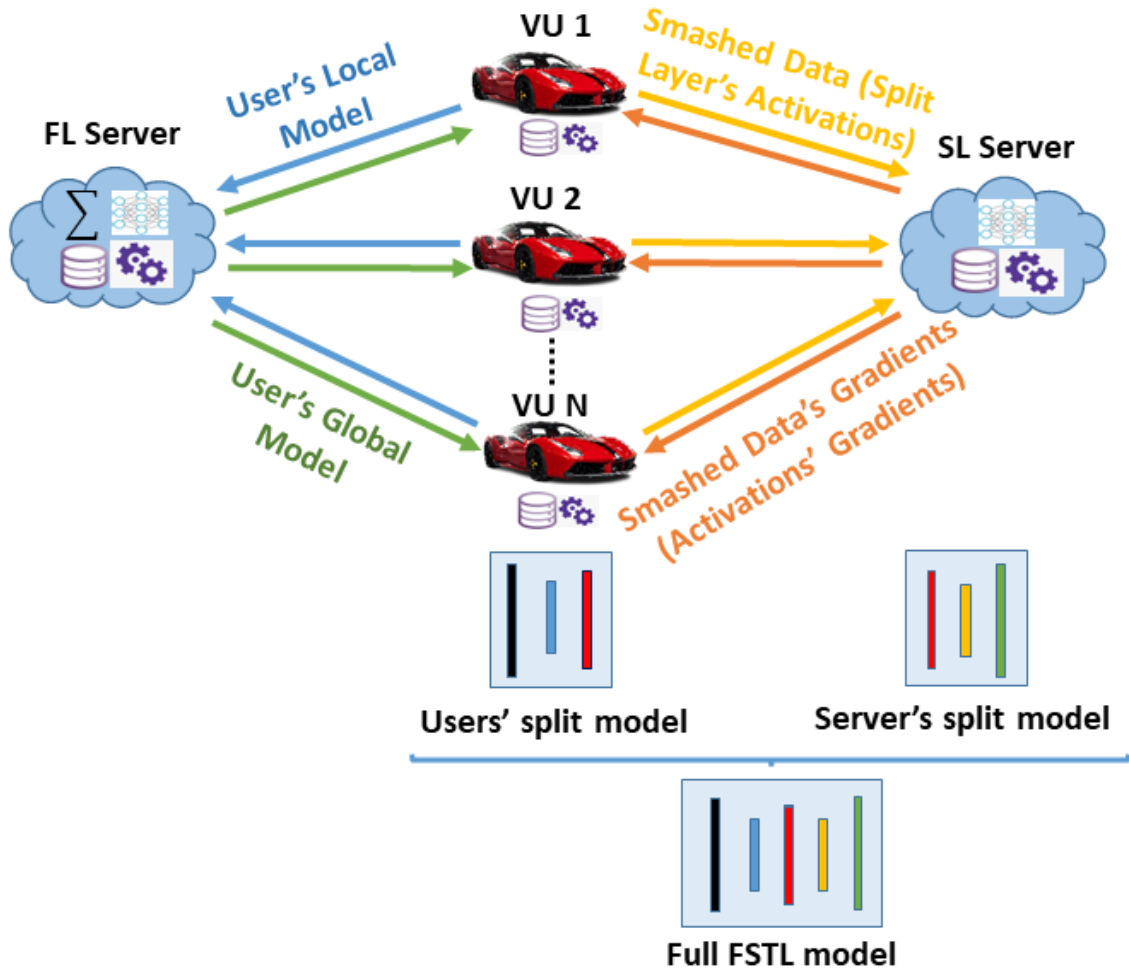


FIGURE 3.1: Overall structure of FSTL

- **Higher Convergence Rate:** By enabling parallel, simultaneous updates from all clients, the model can learn from a diverse set of data within each

training round. This avoids the slow, one-by-one sequential processing inherent in standard SL, leading to a more efficient path to model convergence.

- **Reduced Communication Bottlenecks:** In a sequential learning setup, the server can only communicate with one client at a time, creating a significant bottleneck that scales with the number of users. The FSTL structure removes this limitation by allowing the central server to handle communications with all participating clients in parallel, thereby improving network efficiency.
- **Balanced Computational Loads:** The framework design inherently distributes the workload. While each client handles the initial, less intensive part of the model, the server manages the more computationally demanding tasks and the aggregation of updates from all clients. This parallel structure ensures a more balanced and efficient use of computational resources throughout the network.

### 3.3.2 FSTL Training Process

The FSTL training procedure is an iterative, distributed process that allows multiple users to collaboratively train a model while managing data privacy and distributing computational load. The model is divided into two components: (i) the client-side model, hosted locally by each user, and (ii) the server-side model, hosted centrally.

To formalize the training process, let  $W_U^{p'} = \{\theta_i\}$  denote the parameters of the client-side model and  $W_S^{p'} = \{\theta_s\}$  denote the server-side model parameters. The procedure follows these steps:

1. **Initialization:**

- (a) The global model parameters  $\theta$  are initialized using pre-trained weights from a TL model.
- (b) A specific layer index  $k$  is chosen, determining the split between client and server computations. The layers before  $k$  are hosted on the client side, while the remaining layers are hosted on the server.

2. **Iterative Training:** For each training iteration  $t = 1, 2, \dots, T$ , the following steps occur:

- (a) The initial global model parameters  $\theta$  are distributed to all participating users.
- (b) Each user  $i$  processes its local data  $\mathcal{D}_i$  through its client-split model  $W_U^{p'}$  to generate intermediate representations:

$$H_i = W_U^{p'}(\mathcal{D}_i) \quad (3.7)$$

- (c) The user sends these intermediate representations  $H_i$  to the central server.
- (d) The central server aggregates the intermediate representations from all users by applying a merge function  $\text{Merge}(\cdot)$ :

$$H_{\text{merged}} = \text{Merge}(H_1, H_2, \dots, H_N) \quad (3.8)$$

- (e) Using  $H_{\text{merged}}$ , the server performs a forward-backward pass to compute an updated version of the global model parameters:

$$\theta' = \theta - \eta \cdot \nabla_{\theta} L(H_{\text{merged}}, \theta) \quad (3.9)$$

Here,  $L$  denotes the loss function, and  $\eta$  is the learning rate. This forward-backward process, known as FedAvg, is conducted in a way that preserves privacy since only aggregated updates are calculated.

(f) The central server updates  $\theta$  by setting:

$$\theta = \theta' \quad (3.10)$$

(g) Each user  $i$  then receives the updated global model parameters  $\theta$  from the server and uses them to update their own server-side model parameters:

$$\theta'_i = W_S^{p'}(H_i, \theta) \quad (3.11)$$

(h) Finally, each user  $i$  updates its client-side model parameters:

$$\theta_i = \theta'_i \quad (3.12)$$

This iterative process repeats until the model reaches a stopping criterion, such as convergence of the loss function or a predetermined number of training rounds. In each iteration, users independently process their local data through the client-side model to produce intermediate representations that they then share with the central server. The server merges these representations and performs federated updates to enhance the global model, subsequently sending the updated parameters back to users. This cycle of local data processing, secure sharing, and federated aggregation enables DL while preserving data privacy and distributing computational demands across the network.

### 3.3.3 FSTL Advantages

FSTL offers a robust and efficient framework for DL, combining the strengths of FL, SL, and TL. By allowing users to process data locally while leveraging the global collaborative power of TL-enhanced learning, FSTL improves model performance, speeds up convergence, and minimizes resource constraints, making it an ideal solution for dynamic and privacy-sensitive environments such as ITS [80]. The advantages of FSTL are summarized as follows:

1. **Data Privacy and Security:** By keeping the raw data on each user, FSTL ensures that sensitive information is not shared, addressing privacy concerns inherent in collaborative learning frameworks.
2. **Efficient Communication:** Only the intermediate representations are transmitted between the clients and the server, reducing communication overhead.
3. **Accelerated Convergence:** The use of pre-trained models enables faster convergence, as the models do not need to learn from scratch, but instead build upon previously acquired knowledge.
4. **Improved Performance:** The collaborative learning mechanism at the server level enables global model improvements between all users, allowing the framework to adapt to dynamic environments and various ITS scenarios.

### 3.3.4 FSTL in ITS Scenario

In the context of a 6G-enabled ITS, achieving accurate and adaptive model learning requires diverse sources of data input to reflect the real-world conditions vehicles encounter on roads. In our scenario, we extend FSTL to include not only VUs

but also UAVs and cameras strategically placed on streets and intersections. This diversity of users brings richer environmental data, which is essential for developing responsive and high-performance ITS applications.

VUs provide ground-based data, including real-time information on vehicle location, speed, and road conditions, crucial for direct traffic management and road safety. UAVs, on the other hand, contribute an aerial perspective, capturing broad-area observations that can help monitor larger traffic patterns, assess road congestion, and even support emergency response through quick area scans. Cameras placed on streets and intersections provide fixed high-frequency snapshots of specific intersections or traffic points, enabling an accurate assessment of local traffic density, pedestrian movements, and potential hazards. Together, these users form a comprehensive, multiperspective network that aligns with 6G ITS goals by connecting each ITS infrastructure component.

In this ITS FSTL scenario, illustrated in Figure 3.2, the Road Side Unit (RSU) serves as the FSL server. Here, each type of user (VUs, UAVs, and cameras) retains its raw data locally, thus preserving privacy, and sends only intermediate representations, known as *smashed data*, to the RSU for further processing and model training. This approach significantly reduces communication overhead by limiting exchanged data to model updates, supporting low-latency interaction, and reducing bandwidth demands, important for real-time ITS applications.

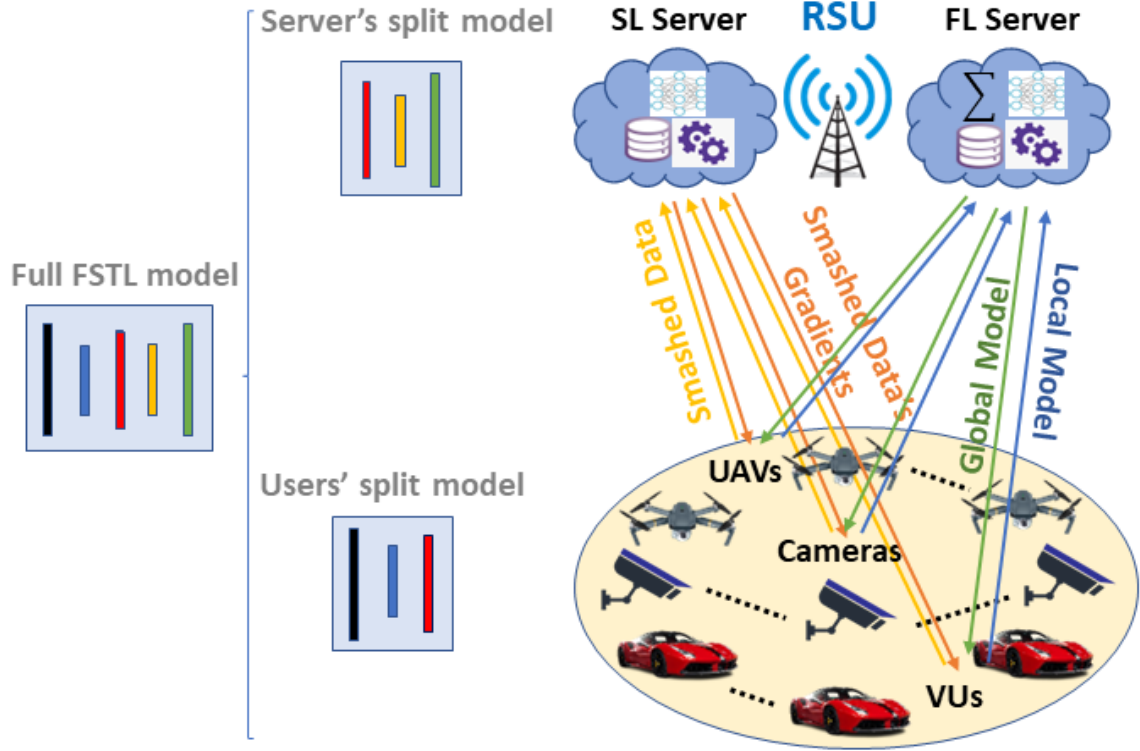


FIGURE 3.2: FSTL structure for ITS with RSU as the FSL Server

**Benefits of FSTL in ITS:** The RSU serves as the main FSL server, which allows collaborative training between VUs, UAVs, and cameras, integrating data from multiple perspectives, and promoting a model that can adapt to varying road conditions. Positioned near traffic points, RSUs are strategically located to provide reliable connection points for each type of user device. They facilitate low-latency communication and rapid data processing, allowing the ITS framework to respond promptly to traffic changes, weather impacts, and other dynamic factors. By aggregating insights from these diverse data sources, the FSTL framework in ITS supports real-time adaptability and robust decision-making, critical for modern transportation systems.

**Training Process:** Each user, in coordination with the RSU, participates in iterative model updates that allow the server to refine the model collaboratively. The



following iterative Algorithm 3.1 outlines this training process. The algorithm begins by initializing the central server's (RSU) model parameters, denoted as  $\theta_s$ , with an initial state  $\theta_s(0)$  (Line 1). Simultaneously, each of the  $N$  individual users initializes their local model parameters,  $\theta_i$ , with their respective initial states  $\theta_i(0)$  (Lines 2-3). This sets up the initial state for both the global and local models before training commences. Each user then independently performs a forward propagation step on their local model using their private local data (Line 4). Instead of sending their raw data or full model, each user computes and transmits an intermediate representation ( $H_i$ ) to the RSU (Line 5). This  $H_i$  likely encapsulates features or activations derived from the local data, without revealing the data itself, thus contributing to data privacy. Upon receiving the intermediate representations ( $H_i$ ) from all active users, the RSU performs a forward propagation step on its own server model with these aggregated representations (Line 6). Subsequently, the RSU computes the gradients with respect to its server model parameters,  $\nabla\theta_s$ , by backpropagating through its model (Line 7). These computed server gradients are then transmitted back to the corresponding users (Line 8). This signifies a collaborative gradient computation where the server contributes to the overall gradient direction. Once users receive the server's gradients ( $\nabla\theta_s$ ), they backpropagate these gradients through their local models (Line 9). This allows them to compute their own local gradients ( $\nabla\theta_i$ ). Finally, each user updates their local model parameters ( $\theta_i$ ) using an optimizer, such as Stochastic Gradient Descent (SGD), with a learning rate  $\eta$  (Line 10-11). The update rule  $\theta_i \leftarrow \theta_i - \eta \cdot \nabla\theta_i$  indicates a standard gradient descent step to minimize a local loss function. After all users have performed their local updates, the RSU aggregates the gradients received from all users. Line 12 describes this aggregation as *Federated Averaging*, where a weighted average of gradients ( $G_{avg} = \frac{1}{N} \cdot \sum(w_i \cdot G_i)$ ) is computed. It is important to note that the algorithm describes the server receiving  $\nabla\theta_s$  back from users in line 8, and then aggregating gradients from all users

in line 13-14. This implies either  $G_i$  refers to  $\nabla\theta_s$  that was sent to users and now is being considered as a contribution, or users are sending back their local gradients  $G_i$  for aggregation. Assuming  $G_i$  here refers to some form of local gradient or update contribution from each user. The RSU then updates its own model parameters ( $\theta_s$ ) using this aggregated gradient, again using a learning rate  $\eta$  (Line 16:  $\theta_s \leftarrow \theta_s - \eta \cdot G_{avg}$ ). This step is crucial for integrating the collective learning from all users into the global model. In the final step of each iteration, the RSU sends its updated server model parameters ( $\theta_s$ ) back to all users (Line 17-18). Each user then synchronizes their local model ( $\theta_i$ ) with the updated server model using a weighted average:  $\theta_i \leftarrow \alpha \cdot \theta_s + (1 - \alpha) \cdot \theta_{avg,i}$ . Here,  $\alpha$  is a hyperparameter that controls the influence of the global server model on the local model, while  $\theta_{avg,i}$  likely refers to the user's previously updated local model, or perhaps an average of their own previous model and their received  $\theta_s$ . This synchronization step ensures that the local models align with the collective knowledge gained by the server, enabling a more robust and generalized model across the distributed system.

This FSTL setup enables real-time, collaborative model improvement across VUs, UAVs, and cameras, strengthening the ITS system's adaptability and performance. The DL approach, anchored by the RSU, balances data privacy and computational efficiency, ensuring that data remains local while intermediate updates support a cohesive, up-to-date model. FedAvg at the RSU uses user-specific weights  $w_i$  to adjust each client's contributions to the global model, enhancing the generalization of the model for varying traffic patterns, weather conditions and environmental factors unique to each type of user.

By leveraging the combined data perspectives from VUs, UAVs, and cameras, FSTL in ITS addresses real-world challenges like traffic flow management, safety monitoring, and emergency response optimization, ensuring robust, privacy-preserving DL

---

**Algorithm 3.1** FSTL Iterative Algorithm for ITS

---

**Input:**  $N, \theta_i(0), \theta_s(0), \alpha, \eta$ **Output:** Updated local and server model parameters  $\theta_i$  and  $\theta_s$ 

- 1: Initialize RSU (server) model parameters:  $\theta_s = \theta_s(0)$
  - 2: **for**  $i = 1$  **to**  $N$  **do**
  - 3:   Initialize user  $i$  with model parameters  $\theta_i = \theta_i(0)$
  - 4:   Perform forward propagation on user model with local data
  - 5:   Send intermediate representations  $H_i$  from user  $i$  to RSU
  - 6:   RSU performs forward propagation on its server model with  $H_i$
  - 7:   Compute gradients  $\nabla\theta_s$  by backpropagating on the RSU model
  - 8:   Transmit gradients  $\nabla\theta_s$  back to user  $i$
  - 9:   Backpropagate on user model using received  $\nabla\theta_s$
  - 10:   Update user local model using an optimizer (e.g., SGD):
  - 11:    $\theta_i \leftarrow \theta_i - \eta \cdot \nabla\theta_i$
  - 12: **end for**
  - 13: RSU aggregates gradients from all users using FedAvg:
  - 14:  $G_{\text{avg}} = \frac{1}{N} \cdot \sum(w_i \cdot G_i)$
  - 15: Update RSU model parameters using aggregated gradients:
  - 16:  $\theta_s \leftarrow \theta_s - \eta \cdot G_{\text{avg}}$
  - 17: Send updated server model parameters  $\theta_s$  back to all users for local model synchronization:
  - 18:  $\theta_i \leftarrow \alpha \cdot \theta_s + (1 - \alpha) \cdot \theta_{\text{avg},i}$
  - 19: **return** Updated parameters  $\theta_i$  for all users and  $\theta_s$  for RSU
- 

that meets the demands of 6G ITS environments.

### 3.4 Generalized Federated Split Transfer Learning

The GFSTL framework extends the FSTL methodology, facilitating scalability across numerous client groups, each group comprising several users. In contrast to the conventional FSTL configuration, which involves training a singular global model for the entire user base, GFSTL segregates the users into discrete groups. This segmentation allows for more adaptable and parallelized model training. This architecture is uniquely suited for complex environments such as NTN, where users are

scattered in various geographical regions and often encounter varying connectivity conditions [77].

In this section, we describe the architecture of GFSTL and the benefits it brings to DL over NTN, particularly in scenarios such as Vehicular Networks (VNs) or EO systems.

### 3.4.1 GFSTL Architecture

The GFSTL architecture, as illustrated in Figure 3.3, begins by partitioning users into  $m$  distinct groups. Each group operates independently with its own subset of users, but all groups work toward training a single global model. Initially, each client (user) is assigned an identical local model, while the server initializes the global model. The server maintains and orchestrates model updates across the groups in parallel, ensuring that the computational load is evenly distributed.

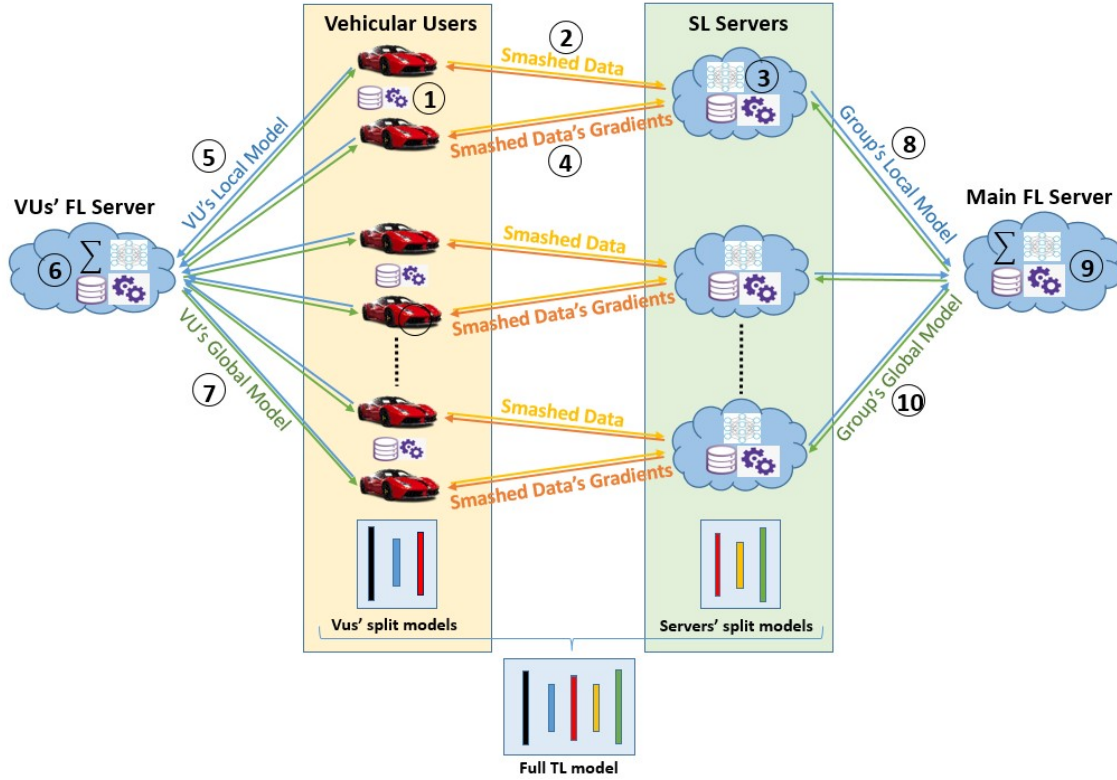


FIGURE 3.3: GFSTL architecture

In Figure 3.3, the architecture of the GFSTL approach is depicted, where each rounded number identifies the corresponding phase of the process. At the beginning of the training process ①, all clients receive a copy of the same initial model. These users perform forward propagation on their local datasets in parallel, generating smashed data, which is the intermediate output from the split NN layers. These smashed data are sent to the SL server associated with each group ②. The server randomly assigns smashed data from each client to one of the predefined groups ③. In each group, forward and backward propagation is performed sequentially within the group, while the groups themselves operate in parallel ④, ensuring efficient use of computational resources. Each SL server (group) produces an updated submodel, which is returned to the corresponding clients for further local backpropagation.

This procedure ensures privacy and efficient use of resources. Privacy is maintained by transmitting only smashed data, which contains no raw information about the local datasets, while computational efficiency is achieved through parallelization at the group level. Gradients from the SL server are applied to the local models, allowing each client to update its parameters based on the feedback received from the server.

Once each group completes its round of FSTL, client-side FL servers aggregate submodels at the group level ⑤-⑦. Subsequently, the main FL server, which could be a central server at a higher-level node, aggregates all the submodels from different groups into a unified global model ⑧-⑩, completing the learning round.

This architecture can accommodate various data configurations, including scenarios with non-iid (non-independent and identically distributed) data, vertically partitioned data, or extended splits across layers. Moreover, the architecture supports different privacy-preserving techniques, as the main server never accesses raw data from the clients, ensuring data confidentiality.

### 3.4.2 GFSTL Training Process

The GFSTL framework, as outlined in Algorithm 3.2, leveraging SL within FL groups. This architecture involves multiple SL servers, each managing a distinct group of users (clients), with a main FL server coordinating across these SL servers for hierarchical aggregation and global model updates. This design aims to enhance privacy, scalability, and efficiency by distributing computational burdens and localizing data. The training process begins with the specified Input parameters:  $N$  (total number of clients), initial client model parameters  $\theta_i(0)$ , initial server model parameters  $\theta_s(0)$ ,  $\alpha$  (synchronization weight), and  $\eta$  (learning rate), and ultimately

produces the Output of updated client and server model parameters  $\theta_i$  and  $\theta_s$ . The training proceeds in an iterative fashion, encompassing nested loops. The outer loop iterates through each of the  $m$  SL server groups, indexed by  $j$  (Line 1). Within this loop, each SL server  $j$  is initialized with its server-side model parameters,  $\theta_{sj}$ , set to their initial state  $\theta_{sj}(0)$  (Line 2). Following this, an inner loop commences for each client  $i$  belonging to group  $j$ , where  $N_j$  is the number of clients in group  $j$  (Line 3). Each client  $i$  initializes its local model parameters,  $\theta_i$ , with pre-trained values  $\theta_i(0)$  (Line 4). The client then performs a forward propagation step on its local model using its private local data (Line 5). Instead of transmitting raw data, the client computes and sends intermediate smashed data ( $H_i$ ) to its corresponding SL server  $j$  (Line 6). Upon receiving  $H_i$  from its clients, the SL server  $j$  performs forward propagation on its server-side model using the received smashed data (Line 7). Subsequently, the SL server computes and performs backpropagation to determine the gradients with respect to its server model parameters,  $\nabla\theta_{sj}$  (Line 8). These computed server gradients,  $\nabla\theta_{sj}$ , are then sent back from the SL server to the respective client  $i$  (Line 9). Upon receiving  $\nabla\theta_{sj}$ , the client  $i$  completes the backpropagation on its own local model using these received gradients (Line 10). After completing backpropagation, the client updates its local model parameters,  $\theta_i$ , using an optimizer, typically Stochastic Gradient Descent (SGD), according to the rule  $\theta_i \leftarrow \theta_i - \eta \cdot \nabla\theta_i$ , where  $\eta$  is the learning rate and  $\nabla\theta_i$  represents the local gradients (Line 11). After all clients in a group have performed their local updates (Line 15 marks the end of the inner loop), the SL server  $j$  aggregates the gradients (or updates) from all clients in its group using federated averaging (Line 12). This produces an aggregated gradient  $G_{avg}$  calculated as  $G_{avg} = \frac{1}{n} \cdot \sum(w_i \cdot G_i)$ , where  $n$  is the number of participating clients in the current aggregation round and  $w_i$  are weights. The SL server then updates its own model parameters,  $\theta_s$ , using an optimizer (e.g., SGD) with the learning rate  $\eta$  and the aggregated gradient  $G_{avg}$ :

$\theta_s \leftarrow \theta_s - \eta \cdot G_{avg}$  (Line 13). Following the SL server model update, each client's local model parameters,  $\theta_i$ , are synchronized with the updated server model using a weighted average:  $\theta_i \leftarrow \alpha \cdot \theta_s + (1 - \alpha) \cdot \theta_{avg,i}$ , where  $\alpha$  controls the influence of the global server model and  $\theta_{avg,i}$  likely refers to the client's previous local model (Line 14). This concludes the operations within each group's inner client loop (Line 15) and the outer group loop (Line 16). After all SL servers have completed their group-level training iterations, they send their updated model parameters to the main FL server (Line 17). The main FL server then aggregates these parameters from all SL servers to form a single, unified global model (Line 18). Finally, this newly formed global model is sent back to all the groups (SL servers and subsequently clients) for synchronization or subsequent training rounds (Line 19). The algorithm then returns the updated local client model parameters  $\theta_i$  for all users and the updated server model parameters  $\theta_s$  for the RSU/main FL server (Line 20).

### 3.4.3 GFSTL Advantages

The GFSTL architecture provides several key advantages in DL systems, especially when applied to NTN:

1. **Scalability:** By distributing clients into multiple groups, the architecture significantly reduces the computational load on any single server. This group-based structure enables scalable training across large numbers of users, each contributing to the global model.
2. **Privacy Preservation:** GFSTL ensures that raw data never leaves the local client devices. Only smashed data, which contains no identifiable information, is transmitted to the SL server, thus preserving the privacy of sensitive user data.



**Algorithm 3.2** GFSTL iterative algorithm**Input:**  $N, \theta_i(0), \theta_s(0), \alpha, \eta$ **Output:**  $\theta_i, \theta_s$ 


---

```

1: for  $1 \leq j \leq m$  do
2:   Initialize SL server  $j$ :  $\theta_{sj} = \theta_{sj}(0)$ 
3:   for  $1 \leq i \leq N_j$  do
4:     Initialize clients:  $\theta_i = \theta_i(0)$ 
5:     Forward propagate on the client model using local data
6:     Send intermediate smashed data ( $H_i$ ) to SL server  $j$ 
7:     Forward propagate on the server model (SL-side model) using  $H_i$ 
8:     Back propagate  $\nabla\theta_{sj}$  on SL server
9:     Send back  $\nabla\theta_{sj}$  from SL server to client $i$ 
10:    Back propagate on the client model using  $\nabla\theta_{sj}$ 
11:    Update local model using an optimizer (e.g., SGD):
        
$$\theta_i \leftarrow \theta_i - \eta \cdot \nabla\theta_i$$

12:    Aggregate (federated average) on SL server:
        
$$G_{avg} = \frac{1}{n} \cdot \sum(w_i \cdot G_i)$$

13:    Update SL server model using an optimizer (e.g. SGD):
        
$$\theta_s \leftarrow \theta_s - \eta \cdot G_{avg}$$

14:    Update client local model parameters:
        
$$\theta_i \leftarrow \alpha \cdot \theta_s + (1 - \alpha) \cdot \theta_{avg,i}$$

15:   end for
16: end for
17: Send model parameters from SL servers to the main FL server
18: Aggregate the parameters to form a global model
19: Send back the global model to the groups
20: return  $\theta_i, \theta_s$ 

```

---

- 3. Efficient Resource Utilization:** The parallelization of groups allows for efficient use of computational resources. The training process within each group is sequential, but groups themselves operate in parallel, which reduces overall training time and enhances performance in large-scale systems.
- 4. Improved Model Accuracy:** By enabling the use of TL with pre-trained models, GFSTL accelerates the convergence of the global model. Moreover,

the aggregation of submodels across groups ensures that the global model is more robust, with improved accuracy due to contributions from diverse groups of users.

5. **Low Communication Overhead:** Since only model parameters are exchanged between the users and the SL servers, the communication overhead is minimal. This is particularly advantageous in NTN, where communication links may be intermittent or costly.

By leveraging these advantages, GFSTL proves to be an efficient, scalable, and privacy-preserving solution for DL in complex environments such as NTN.

### 3.5 Summary of Advantages and Disadvantages of Distributed Learning Techniques

This section synthesizes the key insights from the comparative analysis of DL paradigms presented in this chapter. As summarized in Table 3.2, each technique, i.e., FL, SL, FSL, TL, FSTL, and GFSTL, exhibits a distinct profile of trade-offs between scalability, privacy, computational efficiency, and communication overhead. FL excels in scalability through parallel client training but imposes a high computational burden on clients and significant communication costs for exchanging full model parameters. In contrast, SL drastically reduces client-side computation and enhances privacy by transmitting only intermediate activations (smashed data), but its sequential training nature severely limits scalability and increases latency. FSL emerges as a hybrid solution, balancing the parallelization benefits of FL with the computational and privacy advantages of SL, though it introduces new challenges such as convergence speed and optimal cut-layer selection. The integration of TL

into FSL, resulting in FSTL, further accelerates convergence and improves generalization by leveraging pre-trained models, but requires careful handling of model compatibility and source domain selection. Finally, the GFSTL paradigm introduces a hierarchical, multi-group architecture that offers superior scalability and privacy for large-scale, heterogeneous environments, albeit at the cost of increased system complexity and global coordination latency. This comparative overview underscores that there is no one-size-fits-all solution; the choice of a DL paradigm must be carefully aligned with the specific constraints and objectives of the target application, such as the scale of deployment, device capabilities, network conditions, and stringency of privacy requirements.

### 3.6 Latency Analysis for Multilayer 6G ITS scenario

This section presents a comprehensive analysis of latency across various DL methodologies used in distributed 6G ITS, specifically FL, SL, FSL, FSTL, and GFSTL. In distributed frameworks, the overall latency is influenced by both computation and communication times. For our analysis, we assume that all users—comprising VUs, UAVs, and street-mounted cameras—have uniform data distributions. Including all these client types creates a more realistic ITS scenario, aligning with the 6G vision of interconnected ITS infrastructure.

In the ITS scenario under consideration, the latency is analyzed by defining several key parameters. Let  $d$  denote the total data size being processed, while  $h$  represents the size of the intermediate (smashed) layer output during SL or FSL. The data transmission rate,  $R$ , is a variable dependent on the communication medium. For

TABLE 3.2: Comparison of DL Techniques: Advantages and Disadvantages.

| Technique    | Scalability  | Privacy   | Comp. Efficiency  | Comm. Overhead   |
|--------------|--|---|---|--|
| <b>FL</b>    | + High scalability due to parallel model training across clients.<br>- Limited by full model update aggregation on the server. | + Data remains on local devices (enhances privacy).<br>- Vulnerable to model inversion and poisoning attacks. | - High computation requirement due to full model training on each client.   | - High communication overhead with full model parameter exchange.  |
| <b>SL</b>    | + Suitable for scenarios with fewer clients.<br>- Limited scalability due to serialized client training.                       | + Raw data remains private (smashed data shared only).<br>- Some data leakage can occur via smashed data.     | + Reduced computational burden on client side.<br>- Server-side computation burden remains high.                                | + Lower communication overhead (only smashed data is sent).<br>- High latency due to serial model updates.                 |
| <b>FSL</b>   | + More scalable than SL with parallelized FL elements.<br>- Limited scalability compared to pure FL.                           | + Enhanced privacy through combined FL and SL benefits.<br>- Privacy concerns due to server-client split.     | + Reduced client-side computational load.<br>- Server-side computational load remains significant.                              | + Lower communication overhead compared to FL (smashed data only).<br>- Higher latency than FL due to split training.      |
| <b>TL</b>    | + Applicable for diverse domains; increases scalability in heterogeneous settings.   | + Pre-trained model can mask some sensitive information.  | + Reduces computational load by leveraging pre-trained models.<br>- Limited by domain compatibility of source and target tasks. | + Lower communication overhead due to pre-trained model usage.   |
| <b>FSTL</b>  | + More scalable than FSL, due to faster model convergence using TL.  | + High data privacy with TL masking and combined FL-SL protections.   | + Enhanced efficiency by utilizing pre-trained model splits.  | + Lower communication overhead due to smaller model splits.<br>+ Improved latency by combining parallel FL updates and TL. |
| <b>GFSTL</b> | + Highly scalable due to hierarchical multi-server structure with multiple client groups.                                      | + High data privacy preservation via group separation and SL protections.                                     | + Efficient model convergence with pre-trained models and reduced client computation.   | + Reduced latency by parallel updates within groups and lower bandwidth requirements per user.                             |

training times,  $T$  signifies the time required to train a complete DNN from scratch, whereas  $T'$  and  $T''$  represent training times in FSTL and GFSTL, respectively, utilizing a pre-trained TL model.

Additional factors further impact latency in these frameworks. For instance,  $T_{\text{FedAvg}}$  denotes the time necessary for performing full model aggregation in FL, while  $T_{\text{Merge}}$  represents the merging time of smashed parameters in SL and FSL. Aggregation times are also split into  $T_{\text{UFL}}$ , for handling aggregation on each user group side, and  $T_{\text{MainFL}}$ , for managing aggregation on the main FL server side. Additionally, the full model contains  $q$  parameters, with  $r$  as the ratio of the user-side submodel size to the complete model size, calculated as  $r = \frac{\text{submodel size}}{\text{full model size}}$ .

The total latency is a sum of both computation and communication times:

$$\text{Total Latency} = \text{Total Computation Time} + \text{Total Communication Time} \quad (3.13)$$

Communication latency is particularly significant in DL due to bidirectional transmission requirements. For example, in expressions such as  $2pr$  and  $\frac{2dh}{n}$ , factor 2 reflects the need to upload model updates from users to the server and to download updated parameters back from the server to the clients.

Table 3.3 and the numerical results in Section 4.2.6 demonstrate how latency scales with the number of users  $n$  across these methods. Notably, as  $n$  increases, SL becomes less efficient because its training time is directly proportional to  $n$  due to the serial process. In contrast, FL, FSL, and FSTL methods benefit from parallel training between clients, thus reducing total time even with a larger user base.

In terms of latency ordering:

$$\text{FSTL} < \text{FSL} < \text{FL} < \text{SL}$$

This ordering of latency reflects specific advantages within each framework. FSTL, for instance, achieves lower latency than FSL by integrating TL, which leverages pre-trained models to expedite convergence. This allows FSTL to start from a more informed state, represented by  $T'' < T' < T$ , and ultimately accelerates the training process. FSL also demonstrates greater efficiency over FL because it requires aggregating fewer parameters, making it less computationally demanding; here,  $T_{\text{Merge}} < T_{\text{FedAvg}}$  indicates that merging the smaller model segments in FSL is faster than the full model aggregation in FL. Conversely, SL exhibits the highest latency of these methods due to its reliance on serial processing steps, which prolongs training time and makes it comparatively slower in performance.

In GFSTL, latency is influenced by both the number of users and the distribution between various groups. Let  $n_j$  denote the number of users in the  $j$ -th group. GFSTL latency depends on the slowest group in the system, where communication time is dictated by the group with the greatest delay. To account for this, we employ the  $\max(\cdot)$  operator, which represents the overall latency in GFSTL as:

$$T_{\text{GFSTL}} = T'' + T_{\text{UserFL}} + T_{\text{MainFL}} + \max \left\{ \frac{2dh}{n_j R} \right\} + \frac{2qr}{R} \quad (3.14)$$

Table 3.3 presents summarized latency analysis results for all methods, FL, SL, FSL, FSTL, and GFSTL, illustrating the influence of client numbers  $n$ , data size  $d$ , and communication rate  $R$  on total latency.

In conclusion, this analysis suggests that both FSTL and GFSTL provide substantial latency improvements, especially when scaling to larger user groups within 6G ITS

TABLE 3.3: Latency analysis of five DL methods per round.

| Learning Method | Training and Aggregation Time                 | Comm. user per server   | per per | Total comm. per server | Total comm. time  | Total latency   |
|-----------------|---|-------------------------|---------|------------------------|---|---|
| FL              | $T + T_{\text{FedAvg}}$                       | $2q$                    |         | $2nq$                  | $\frac{2q}{R}$  | $T + T_{\text{FedAvg}} + \frac{2q}{R}$  |
| SL              | $T$   | $\frac{2dh}{n} + 2qr$   |         | $2dh + 2nqr$           | $\frac{2dh}{R} + \frac{2nqr}{R}$                          | $T + \frac{2dh}{R} + \frac{2nqr}{R}$  |
| FSL             | $T + T_{\text{Merge}}$                        | $\frac{2dh}{n} + 2qr$   |         | $2dh + 2nqr$           | $\frac{2dh}{nR} + \frac{2qr}{R}$                          | $T + T_{\text{Merge}} + \frac{2dh}{nR} + \frac{2qr}{R}$   |
| FSTL            | $T' + T_{\text{Merge}}$                       | $\frac{2dh}{n} + 2qr$   |         | $2dh + 2nqr$           | $\frac{2dh}{nR} + \frac{2qr}{R}$                          | $T' + T_{\text{Merge}} + \frac{2dh}{nR} + \frac{2qr}{R}$  |
| GFSTL           | $T'' + T_{\text{UserFL}} + T_{\text{MainFL}}$ | $\frac{2dh}{n_j} + 2qr$ |         | $2dh + 2n_jqr$         | $\max \left\{ \frac{2dh}{n_j R} \right\} + \frac{2qr}{R}$ | $T'' + T_{\text{UserFL}} + T_{\text{MainFL}} + \max \left\{ \frac{2dh}{n_j R} \right\} + \frac{2qr}{R}$ |

scenarios. These gains stem from pre-trained model usage and efficient parameter aggregation, outperforming conventional FL and SL approaches in latency-sensitive applications.

### 3.7 Multilayer Distributed GFSTL Scenarios

Having established the theoretical foundations of the FSTL and GFSTL frameworks, this section transitions to their practical application within the complex, multilayered 6G ecosystem. The true potential of a scalable, hierarchical framework like GFSTL is realized when it is deployed across integrated T/NTNs to solve real-world problems. To demonstrate this versatility, we will first explore two distinct, high-impact use cases: a GFSTL-based architecture for vehicular AGIN and a configuration for NTN-based EO. We will then present a unified architecture that synthesizes these scenarios, providing a comprehensive solution for next-generation ITS, and conclude with a formal latency analysis of this integrated system.

### 3.7.1 Multilayer GFSTL in Vehicular Aerial-Ground Integrated Network

Aerial networks have gained significant attention as a viable solution to extend connectivity and support advanced applications in complex and challenging environments. These networks are particularly beneficial for vehicular scenarios, which are characterized by high mobility, inconsistent connectivity, and dynamic data distribution—factors that pose substantial challenges to traditional ML methods [111]. In our previous work[80], the concept of FSTL was introduced to address these challenges in vehicular scenarios. Building upon this, the current study extends the applicability of FSTL by integrating it with aerial networks to establish a unified Aerial-Ground Integrated Network (AGIN) for vehicular scenarios. This framework allows for the full exploitation of the GFSTL paradigm.

By leveraging HAPs and the existing capabilities of FSTL, we introduce GFSTL to harness the advantages of aerial networks, enabling efficient, scalable, and secure training and inference processes for VUs. The proposed AGIN architecture is designed to overcome the limitations of ground-based networks by integrating aerial nodes, enhancing connectivity, reducing latency, and maintaining data privacy.

#### 3.7.1.1 System Architecture and Methodology

In the proposed scenario shown in Figure 3.4, RSUs function as both SL and FL servers for VUs, facilitating the execution of FSTL. Meanwhile, HAPs serve as the main FL servers, overseeing the aggregation of models across multiple RSUs. This architecture enables SL to operate locally on the VUs, ensuring that raw data remains on the devices, thus safeguarding privacy. At the same time, more computationally intensive tasks, such as model training, are offloaded to HAPs. This



distributed approach significantly reduces communication overhead, as only model updates—rather than raw data—are transmitted between VUs and HAPs.

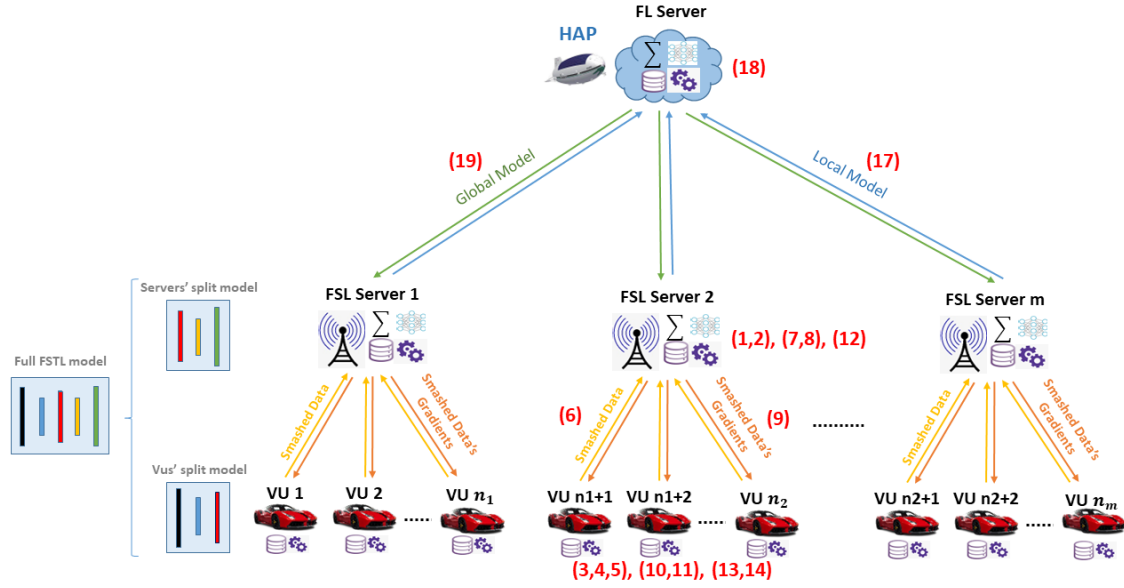


FIGURE 3.4: GFSTL structure for vehicular AGIN.

The FL paradigm in this setup enables collaborative training across multiple VUs, promoting knowledge sharing and allowing the models to adapt to diverse vehicular environments. Each RSU is responsible for aggregating the updates of its local group of VUs, while the HAP performs a higher-level aggregation, merging model updates from all RSUs into a unified global model.

### 3.7.1.2 Advantages of Integrating GFSTL with Aerial Networks

The integration of FSTL with HAPs unlocks several new possibilities for vehicular scenarios. First, the high-altitude placement of HAPs ensures extensive coverage, reducing the likelihood of interference and maintaining seamless connectivity for VUs, even in remote or underserved areas where terrestrial infrastructure may be limited or unavailable. Second, the FL architecture enhances collective intelligence across

VUs by enabling them to collaboratively train models that are better adapted to the dynamic conditions and varying vehicular scenarios encountered on the ground.

Through the AGIN framework, the combination of RSUs and HAPs offers enhanced connectivity, greater data privacy, and improved decision-making capabilities. Using both ground and aerial elements, our proposed approach overcomes the connectivity limitations inherent in traditional VNs, providing reliable and scalable solutions to support intelligent VNs.

### 3.7.1.3 GFSTL Workflow in Vehicular Scenarios

In the vehicular AGIN scenario, GFSTL is employed to enable collaborative learning across multiple vehicles, while ensuring data privacy throughout the process. Each RSU acts as the FSL server for its designated group of VUs, handling model updates and exchanging gradients with the VUs. Once each group completes the FSTL training process, the aggregated FSTL models are sent to the HAP, which serves as the main FL server, to perform a final aggregation across all groups.

Specifically, the workflow can be described as follows:

1. The RSUs first act as local FSTL servers, managing the training and model aggregation for their respective groups of VUs.
2. Each VU performs local computation and forwards its model updates to its corresponding RSU, which aggregates these updates.
3. Once the FSTL process is completed for all groups of VUs managed by the RSUs, the fully trained FSTL models are transmitted from the RSUs to the HAP.

4. The HAP, functioning as the main FL server, aggregates the complete set of model parameters from all RSUs, thereby finalizing the global model.

This architecture not only optimizes the training process by distributing the computation and aggregation tasks between RSUs and HAPs but also preserves privacy by ensuring that raw data never leaves the local VUs.

### 3.7.2 Multilayer GFSTL in NTN-based EO

In this section, we introduce the proposed GFSTL process for EO applications, along with its key advantages and the challenges it seeks to address. This generalized architecture, illustrated in Figure 3.5, is tailored for multilayer EO applications. This framework integrates NTNs, including Low-Altitude Platforms (LAPs), HAPs, and Low Earth Orbit (LEO) satellites, into the GFSTL architecture, leveraging their distinct advantages to enhance EO data collection, processing, and analysis.

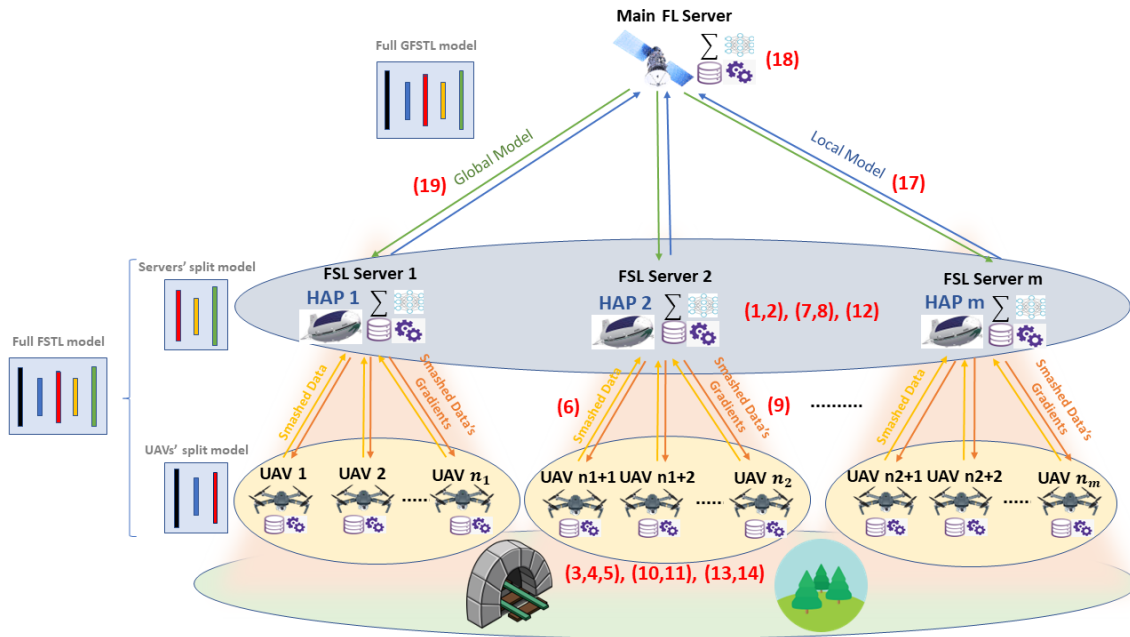


FIGURE 3.5: GFSTL structure for Earth Observation using Non-Terrestrial layers

### 3.7.2.1 Motivation and Proposed Framework

The growing need for accurate and high-resolution EO data in applications such as environmental monitoring, climate change analysis, and disaster response presents several challenges, particularly in terms of data collection, processing, and transmission. Traditional EO systems, heavily reliant on satellite networks, often suffer from latency, data security concerns, and limited accuracy in capturing information in dense or remote environments (e.g., forests, and urban areas) [79].

Our proposed framework addresses these limitations by optimally combining the precision and task-specific capabilities of Unmanned Aerial Vehicles (UAVs) with the extended coverage and computational power of HAPs. UAVs are ideal for collecting high-precision data in specific areas, especially in challenging environments, while HAPs provide broader coverage and handle computationally intensive tasks. By integrating these NTN into the GFSTL architecture, our framework ensures efficient and secure DL across multiple EO platforms.

This framework addresses the first major gap in the literature by enabling optimal integration of UAVs, HAPs, and LEO satellites to enhance accuracy, coverage, and resource efficiency in EO applications. Furthermore, our framework introduces an efficient DL method through GFSTL, which combines the strengths of FL for secure model training, TL to expedite the training process, and SL to ensure that resource-constrained UAVs can participate effectively in the learning process.

### 3.7.2.2 Framework Components and Advantages

In the proposed framework, HAPs function as FSL servers, which integrate both SL and FL server functionalities. This allows UAVs to retain raw, high-quality data locally while offloading computationally intensive tasks to HAPs. This setup ensures

privacy preservation and reduces the bandwidth required for transmitting large volumes of data since only intermediate outputs (smashed data) are transmitted.

LEO satellites serve as the central main FL servers, orchestrating collaborative model training across multiple HAP-UAV groups. This architecture fosters collective intelligence capable of adapting to various EO scenarios. Using the hierarchical structure of NTN, the framework optimizes resource utilization while ensuring scalability and adaptability to different EO tasks, such as image analysis, clustering, and object recognition.

The proposed framework brings numerous advantages:

1. **Increased Accuracy:** UAVs can focus on capturing precise data in specific regions, especially in areas where satellite imagery might be less effective (e.g., dense forests). This localized data collection ensures higher accuracy and relevance of EO data.
2. **Resource Efficiency:** HAPs serve as intermediaries between UAVs and LEO satellites, efficiently distributing computational tasks and enabling seamless data processing across layers.
3. **Privacy and Security:** The FSL approach ensures that raw data remains on local devices (UAVs), preserving privacy while transmitting only intermediate data (smashed data) to the HAP servers. This minimizes the risk of data breaches and ensures compliance with privacy regulations.
4. **Reduced Latency:** By utilizing TL to initialize the training process with pre-trained models, the framework reduces the total training time, enabling faster convergence of the global model.

### 3.7.2.3 GFSTL Training Process for EO over NTN

The GFSTL process over NTNs for EO applications is described in Algorithm 3.3. The process begins by initializing the UAV models and server models (parameters  $\theta_i$  and  $\theta_s$ ), which are pre-trained on a DNN (such as ResNet). Each UAV performs forward propagation using its local data, sending the intermediate outputs (smashed data) to the HAP server for further processing. The HAP server computes gradients for the server model parameters and sends them back to the UAVs, allowing for backpropagation and local model updates. For better understanding, the number in Figure 3.5 refers to the step individualized by the lines of the Algorithm 3.3.

---

**Algorithm 3.3** GFSTL iterative algorithm over NTN for EO
 

---

- 1: **for**  $1 \leq j \leq m$  **do**
  - 2:   Initialize FSL server  $j$ :  $\theta_{sj} = \theta_{sj}(0)$
  - 3:   **for**  $1 \leq i \leq n_j$  **do**
  - 4:     Initialize  $UAV_i$ :  $\theta_i = \theta_i(0)$
  - 5:     Forward propagate on  $UAV_i$  model using local data
  - 6:     Send intermediate smashed data ( $H_i$ ) to  $HAP_j$
  - 7:     Forward propagate on  $HAP_j$  model using  $H_i$
  - 8:     Back propagate  $\nabla\theta_{sj}$  on  $HAP_j$  model
  - 9:     Send back  $\nabla\theta_{sj}$  from  $HAP_j$  to  $UAV_i$
  - 10:    Back propagate on  $UAV_i$  model using  $\nabla\theta_{sj}$
  - 11:    Update  $UAV_i$  local model using an optimizer (e.g., SGD)
  - 12:    Aggregate (federated average) on  $HAP_j$ :
 
$$G_{avg} = \frac{1}{n} \cdot \sum(w_i \cdot G_i)$$
  - 13:    Update  $HAP_j$  model using an optimizer (e.g., SGD)
  - 14:    Update  $UAV_i$  local model parameters
  - 15:   **end for**
  - 16: **end for**
  - 17: Send model parameters from HAP-UAV groups to LEO satellite
  - 18: Aggregate the parameters to form a global model
  - 19: Send back the global model to the groups
- 

### 3.7.2.4 Added Benefits and Challenges

The GFSTL framework brings significant improvements to EO applications by addressing key gaps in data collection, processing, and learning efficiency. The use of

UAVs for precision data capture improves the quality of images and data collected, especially in challenging terrains. The integration of HAPs and TL accelerates the training process, optimizing computational resources and improving model convergence.

However, several challenges need to be addressed for the successful deployment of GFSTL in EO systems. One major challenge is task compatibility; choosing a pre-trained model that aligns well with the target domain is critical to avoid domain shift and model drift. Additionally, scalability remains a concern, especially as more UAV-HAP groups are added. These challenges must be carefully considered to ensure optimal performance.

### **3.8 Unified Hierarchical GFSTL for 6G ITS**

The 6G era envisions a fully connected world where all elements of the ITS infrastructure are seamlessly integrated, from vehicles to aerial systems and ground-based sensors. This hyper-connectivity provides the foundation for real-time data processing and intelligent decision-making, essential to modern transportation systems. A hierarchical, multilayer architecture, such as the AGIN or NTN, is ideally suited to support the demands of 6G ITS, where the massive scale, low latency, and need for privacy-preserving DL are critical factors. This section introduces the concept of multilayer GFSTL in the context of 6G ITS, providing a flexible, scalable, and efficient solution to the challenges posed by high data volumes and complex model training scenarios.



### 3.8.1 Architecture and Components

In the 6G ITS ecosystem, all ITS infrastructure components—including VUs, UAVs, and ground-based Cameras (strategically positioned at intersections, crossroads, and high-traffic areas)—are interconnected to enhance safety, efficiency, and overall traffic management. This requires a hierarchical, multilayer system where computational tasks are distributed across different network layers, improving both resource management and real-time decision-making. The architecture consists of three main components:

1. **Users (VUs, UAVs, and Cameras):** VUs are vehicles equipped with various sensors and communication modules that gather real-time traffic and environmental data. UAVs provide additional data, particularly in areas that are difficult to monitor or reach from the ground, offering support in vehicle navigation, surveillance, and emergency response. Street cameras positioned at critical points such as intersections and crossroads monitor traffic flow, detect accidents, and feed real-time data into the network.
2. **FSL Servers (RSUs and Base Stations (BSs)):** RSUs and BSs act as FSL servers. They aggregate and process data received from nearby VUs, UAVs, and cameras, performing preliminary computations.
3. **Main FL Server (HAP):** A HAP serves as the main FL server. It receives and aggregates the models trained by the FSL servers (RSUs/BSs), conducting the final stage of training which is aggregation to a global model that integrates data from all layers of the ITS system.

The hierarchical structure, depicted in Figure 3.6, enables efficient model training through GFSTL, where the learning tasks are split across multiple layers—ground-based (VUs, RSUs, and BSs) and aerial (UAVs, HAPs). This ensures both real-time data analysis and long-term predictive modeling, addressing the dynamic requirements of 6G ITS.

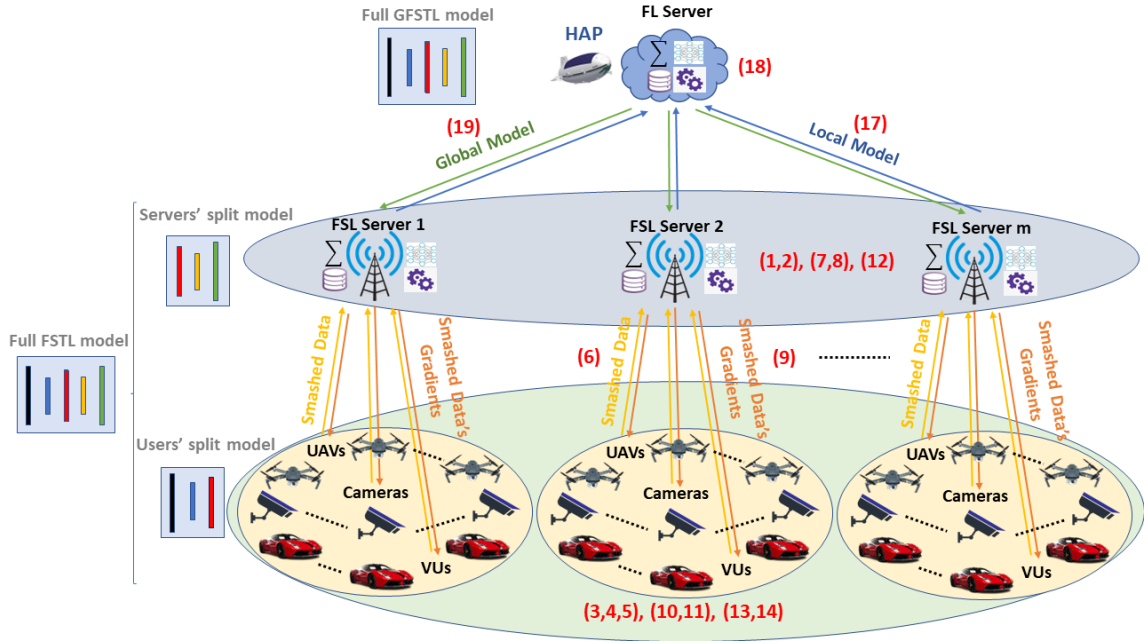


FIGURE 3.6: Multilayer GFSTL Architecture for 6G ITS

### 3.8.2 Training Process and Iterative Algorithm

In the multilayer GFSTL for 6G ITS, the training is performed iteratively as in Algorithm 3.4, with each layer contributing to the global model through a combination of SL, FL and TL. The process is as follows:

---

**Algorithm 3.4** GFSTL Iterative Algorithm for 6G ITS
 

---

- 1: Initialize global model  $\theta_s = \theta_s(0)$
- 2: **for** each RSU/BS  $j = 1$  to  $m$  **do**
- 3:     Initialize FSL server  $j$ :  $\theta_{sj} = \theta_{sj}(0)$
- 4:     **for** each client  $i = 1$  to  $N_j$  **do**
- 5:         Forward propagate on client model  $\theta_i$  using local data
- 6:         Send intermediate smashed data to FSL server  $j$
- 7:         Forward propagate on FSL server model  $\theta_{sj}$  using smashed data
- 8:         Compute gradients  $\nabla\theta_{sj}$  on server model
- 9:         Send gradients  $\nabla\theta_{sj}$  back to client
- 10:        Backpropagate on client model  $\theta_i$
- 11:        Update client model using optimizer (e.g., SGD):

$$\theta_i \leftarrow \theta_i - \eta \nabla\theta_i$$

- 12:        Aggregate gradients on FSL server  $j$ :

$$G_{avg} = \frac{1}{N_j} \sum (w_i \cdot G_i)$$

- 13:        Update FSL server model  $\theta_{sj}$
  - 14:     **end for**
  - 15: **end for**
  - 16: Send updated models from FSL servers to main FL server (HAP)
  - 17: Aggregate global model at HAP:  $\theta_s$
  - 18: Redistribute updated global model to all clients
- 

1. **Initialization:** Each client (VUs, UAVs, Cameras) is initialized with a pre-trained model that is specific to its task, with parameters denoted as  $\theta_i$  for the

client model and  $\theta_s$  for the server model.

2. **Local Model Update:** Clients perform forward propagation on their local models using the collected data, producing intermediate results (smashed data) that are sent to the nearest FSL server (RSU or BS).
3. **FSL Server Aggregation:** RSUs and BSs aggregate the intermediate results, perform forward and backward propagation on their server models, and send the corresponding gradients back to the clients to update their local models.
4. **Main FL Server Aggregation:** Once all FSL servers complete their local training, they send the updated models to the main FL server (HAP), aggregating these models into a unified global model.
5. **Global Model Update:** The global model is redistributed to the clients through the FSL servers for the next training iteration.

The iterative process outlined above ensures that all 6G ITS infrastructure components collaborate to build a unified, intelligent model. This model adapts to dynamic traffic conditions and learns from various data sources.

### 3.8.3 Added Benefits and Challenges

In the following, we delve into the core aspects of the proposed GFSTL framework, outlining its significant advantages and the inherent complexities that need to be addressed. The unique architecture of GFSTL offers a robust solution for ITS while also presenting several considerations for its optimal implementation.

**Benefits:** The GFSTL framework is designed to deliver a multitude of advantages that enhance the efficiency, privacy, and reliability of ITS. These benefits stem from

its innovative multilayered and distributed approach that addresses key limitations of traditional centralized systems.

- **Scalability:** Multilayer GFSTL is inherently scalable, allowing it to accommodate the increasing number of VUs, UAVs, and cameras as the ITS infrastructure grows. Its distributed nature ensures that the computational load is balanced throughout the system.
- **Privacy Preservation:** Since raw data remain on the client side (VUs, UAVs, Cameras), the model leverages FSL to ensure privacy-preserving model training, which is crucial for sensitive traffic data and user privacy.
- **Reduced Latency:** By distributing computational tasks across FSL servers (RSUs/BSs) and the main FL server (HAP), GFSTL minimizes communication overhead and reduces the time needed to update the global model.
- **Resilience:** The integration of multiple hierarchical layers ensures redundancy, making the system more resilient to failures in individual components, such as RSUs or UAVs.

**Challenges:** Despite its promising benefits, the deployment of a sophisticated system like GFSTL in a 6G ITS environment comes with its own set of challenges. These complexities arise from the distributed nature of the system, the diversity of data sources, and the need for careful resource orchestration to ensure optimal performance and model accuracy.

- **Resource Management:** Efficiently managing network and computation resources across VUs, UAVs, cameras, RSUs, and the HAP is critical. Bandwidth Allocation (BA), processing power, and memory management are areas that need careful optimization.

- **Data Heterogeneity:** Data collected from various components (VUs, UAVs, and Cameras) may differ in format and quality. Ensuring that the global model can generalize across heterogeneous data sources is a challenging task.
- **Model Convergence:** Although TL and SL help to accelerate the training process, achieving convergence in a system as complex as 6G ITS requires fine-tuning of hyperparameters, especially with regard to the learning rate and aggregation intervals.

### 3.9 Simulations and Performance Evaluations

The proposed GFSTL method for 6G ITS is evaluated through simulations performed on a Python-based platform, using libraries such as Pandas, NumPy, and Matplotlib for data processing and visualization. To accelerate training and handle high-dimensional data typical of ITS applications, we employ NVIDIA<sup>®</sup> Tesla<sup>®</sup> T4 GPU accelerators.

Given the requirements for real-time object detection and situational awareness in ITS, we employ one of the latest versions of YOLOv5 as our base model [112]. YOLOv5, which is optimized for both accuracy and speed in object detection, is designed to rapidly process complex urban scenes, enabling it to detect multiple objects such as vehicles, pedestrians, traffic lights, and signs within a single frame. This makes it well-suited for ITS applications where rapid detection and categorization of diverse objects are essential for tasks such as traffic monitoring, congestion management, and accident prevention.

Furthermore, we utilize the Cityscapes dataset [113] for training and testing. This dataset, which contains more than 5000 high-resolution images from urban street

environments, provides dense pixel-level annotations across multiple object classes pertinent to ITS, including vehicles, road markings, pedestrians, and environmental elements. Unlike simpler datasets such as MNIST, Cityscapes represents real-world complexities in urban environments with variable lighting, weather, and dynamic backgrounds, making it more suitable for evaluating model robustness in ITS applications.

To comprehensively evaluate GFSTL, we analyze its performance using key metrics such as model accuracy, latency, communication efficiency, and scalability. Simulations are carried out in a multilayer ITS setup, where each RSU serves as an FSL server connected to a batch of five VUs, two UAVs, and three stationary cameras positioned at intersections or crossroads to provide diverse data inputs. Three RSUs act as distributed FSL servers, while a HAP functions as the main FL server responsible for aggregating models received from each RSU. This multilayer configuration reflects a hierarchical AGIN structure suitable for 6G ITS, where every component of the infrastructure is interconnected to optimize data processing and model performance.

### 3.9.1 Simulation, Hardware, and Network Parameters

Table 3.4 reports the exact experimental configuration used throughout this section, including the dataset splits, hardware, network assumptions, and split-learning choices. For object detection, we employ the YOLOv5-m variant (Ultralytics): approximately  $21.2 \times 10^6$  parameters and  $\approx 49$  GFLOPs at an input resolution of  $640 \times 640$  [114]. This model was selected as a practical trade-off between detection accuracy and inference cost for ITS use cases, where RSU/HAP servers operate with

Tesla T4 accelerators (16 GB GDDR6) and edge clients have limited memory/compute budgets [115].

The main training and per-client hyperparameters used in all experiments are as follows: input size =  $640 \times 640$ , per-client local batch size = 8, optimizer = Adam, initial learning rate =  $1 \times 10^{-4}$ , weight decay =  $1 \times 10^{-4}$ , and local epochs per round = 1. The Cityscapes dataset splits follow standard practice (Train = 2975; Val = 500; Test = 1525) [113]. The per-batch composition in our ITS scenarios is five VUs, two UAVs, and three street cameras (one batch), and experiments sweep from 1 to 10 batches per RSU.

To support Split/Transfer Learning, we cut the network at the last neck layer (just before the detection head) and transmit a compact “smashed” feature vector per image. The smashed representation used in our experiments is 256 floats (single-precision), i.e.,

$$\text{smashed size per image} = 256 \times 4 \text{ bytes} = 1024 \text{ bytes} \approx 1.0 \text{ KB.}$$

With batch size = 8, the per-client smashed payload per local update is

$$8 \times 1.0 \text{ KB} = 8,192 \text{ bytes} = 65,536 \text{ bits.}$$

Under the baseline link rate of 100 Mbps (that is,  $100 \times 10^6$  bits/s), the raw transmission time for this payload (neglecting protocol overhead and RTT) is

$$\frac{65,536 \text{ bits}}{100 \times 10^6 \text{ bits/s}} = 6.5536 \times 10^{-4} \text{ s} \approx 0.66 \text{ ms.}$$



By contrast, uploading the full YOLOv5-m model as in standard FL would transfer approximately

$$21.2 \times 10^6 \text{ params} \times 4 \text{ bytes/param} = 84,800,000 \text{ bytes} \approx 84.8 \text{ MB},$$

which is impractical for per-round edge uploads on the considered link rates. These arithmetic examples motivate why GFSTL transmits compact smashed representations (KB-scale per image) rather than full models (tens of MB), reducing the per-round communication burden by orders of magnitude.

Table 3.4 lists all the values of the hardware and network parameters used as a baseline for the latency and communication calculations presented in Section 3.6 and the comparisons of the methods later in this section. Using the same physical configuration for FL, SL, FSL, FSTL, and GFSTL ensures that the differences reported later in Table 3.5 of Section 3.9.5 (final mAP, rounds to converge, end-to-end latency, and per-round communication volume) arise from the learning paradigms themselves rather than from hardware or network discrepancies, allowing a fair and reproducible comparison.

### 3.9.2 Model Convergence and Accuracy versus Rounds

To assess the convergence behavior of the proposed frameworks, we evaluated model accuracy across training rounds and compared it with three baseline DL paradigms: FL, SL and FSL. All methods use the same dataset (Cityscapes), with YOLOv5 as the base model [99], and clients are initialized with Transfer Learning from a pre-trained YOLOv5 checkpoint. The results are shown in Figure 3.7.

TABLE 3.4: Simulation, hardware, and network parameters.

| Parameter                            | Value/Type   |
|--------------------------------------|--|
| YOLOv5 variant                       | YOLOv5-m (21.2 M params, 49 GFLOPs @ $640 \times 640$ ).                                       |
| Input image size                     | $640 \times 640$ px.   |
| Pre-trained weights                  | YOLOv5 (Ultralytics) pre-trained then fine-tuned on Cityscapes.                                |
| Cityscapes images (fine annotations) | Train: 2975; Val: 500; Test: 1525.   |
| Per-batch composition                | 5 VUs, 2 UAVs, and 3 static cameras (per batch).   |
| Max batches per RSU                  | 1–10 (experiments sweep).  |
| GPU                                  | NVIDIA Tesla T4, with 16 GB of GDDR6.  |
| RSU compute                          | $1 \times$ Tesla T4 (per RSU experiment).  |
| HAP compute (main FL server)         | $2 \times$ Tesla T4 (simulated higher-tier server).  |
| CPU (host)                           | Intel Xeon 8 cores @ 2.3 GHz (typical server host).  |
| Per-client memory (simulated)        | 4 GB (typical VU/UAV/camera edge device budget).   |
| Batch size (per client)              | 8 images.  |
| Optimizer                            | Adam, initial LR = $1 \times 10^{-4}$ , weight decay = $1 \times 10^{-4}$ .                    |
| Training epochs per local update     | 1 (local)/global rounds up to 200 (early stopping).  |
| Communications link rate (R)         | 100 Mbps uplink/downlink.  |
| Smashed representation (cut layer)   | 256-dim float vector per image ( $256 \times 4$ bytes = 1.0 KB).                               |
| Smashed payload per batch            | $1.0 \text{ KB} \times \text{batch size (8)} \times \text{number of clients per batch (10)}$ . |

As seen in the figure, the GFSTL method achieves the highest initial accuracy ( $\approx 0.956$  at round 1) and converges rapidly toward near-optimal performance ( $\approx 0.998$  by round 8). In contrast, FSTL starts at  $\approx 0.935$  and stabilizes around 0.958 after 10 rounds, while FL and FSL show slower and less stable growth, plateauing near 0.951 and 0.940, respectively. SL performs the weakest overall, starting near 0.920 and reaching only  $\approx 0.933$  at round 10. These numerical results confirm the clear ordering of  $\text{GFSTL} > \text{FSTL} > \text{FL} > \text{FSL} > \text{SL}$  in terms of both convergence speed and final accuracy.

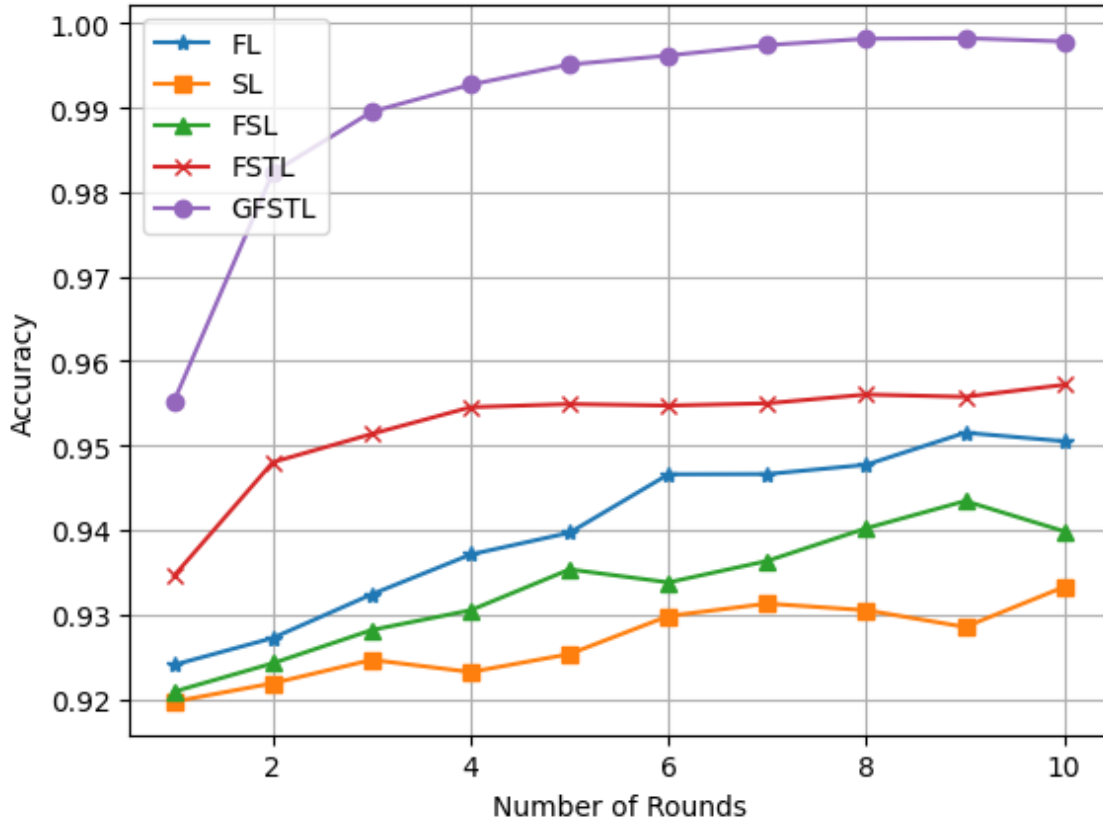


FIGURE 3.7: Accuracy of different DL methods vs. number of training rounds.

The superior performance of GFSTL can be explained by three factors. First, **accelerated knowledge transfer** provides all clients with pre-trained feature representations, allowing GFSTL to start from a higher baseline accuracy than the other methods. Second, **hierarchical synchronization between the FL and SL servers** allows more frequent and efficient updates, which reduces drift between heterogeneous clients and shortens the number of rounds needed for convergence. Third, **flexible aggregation across RSUs and HAPs** in GFSTL leverages the multilayer T/NTN architecture, which improves scalability and robustness when handling diverse vehicular, UAV, and roadside camera data sources.

It should also be noted that the accuracy of GFSTL quickly saturates after round 6, indicating efficient utilization of the available data and suggesting that relatively

few communication rounds are needed to achieve near-optimal performance. This has direct implications for latency and bandwidth usage in ITS scenarios, where minimizing the number of communication rounds is crucial. In contrast, FL, SL, and FSL require many more rounds to approach their respective plateaus, which implies higher communication costs and less efficient use of network resources.

### 3.9.3 User Diversity and Model Accuracy

To evaluate the robustness of the proposed frameworks under varying levels of user diversity, we simulate scenarios in which the number of user batches per RSU is increased from 2 to 10. Each batch contains five VUs, two UAVs, and three cameras, thus capturing a wide range of perspectives and sensing modalities. All approaches utilize YOLOv5, which is initialized with a pre-trained Cityscapes model, to ensure a fair baseline. Figure 3.8 presents the resulting accuracies for GFSTL, FSTL, FSL, FL, and SL.

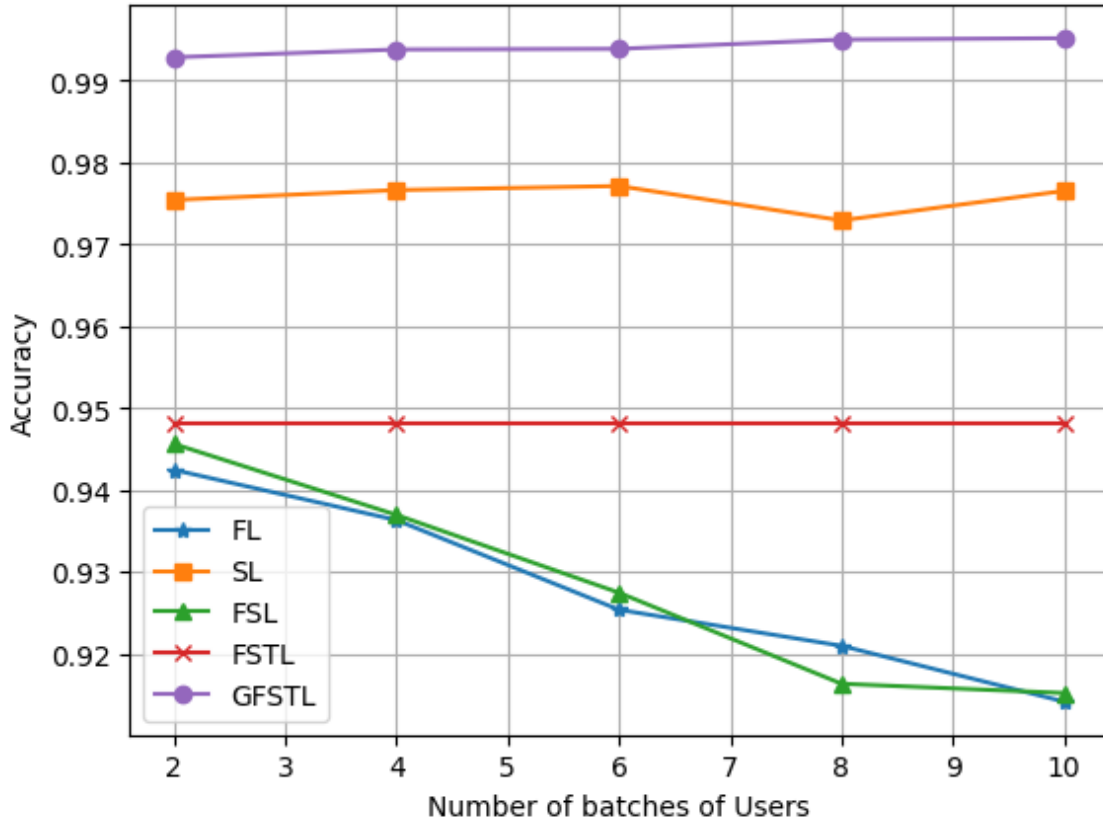


FIGURE 3.8: Accuracy of different DL methods vs. number of user batches per RSU.

The results show a clear separation in performance. GFSTL consistently maintains near-constant high accuracy ( $\approx 0.992$ – $0.997$ ) across all batch sizes, demonstrating its scalability and resilience to the challenges posed by heterogeneous data sources. FSTL achieves stable accuracy around 0.949 but does not improve with additional user batches, suggesting that while its hybrid structure reduces variance, it lacks the flexibility of the hierarchical server placement and transfer initialization of GFSTL. In contrast, FL and FSL exhibit a marked degradation in accuracy as the number of batches increases: from  $\approx 0.944$  at 2 batches to below 0.916 at 10 batches. This drop highlights the difficulty of handling statistical heterogeneity and non-i.i.d. data distributions when only parameter averaging or basic split mechanisms are applied. SL performs better than FL and FSL, stabilizing around 0.976, but it falls short of

GFSTL because it lacks federated aggregation and thus struggles to fully generalize across diverse users.

Two insights emerge from these observations. First, **GFSTL’s knowledge transfer via pre-trained initialization** enables each client to begin with strong representations of urban scene features, ensuring that even highly diverse client data can be quickly adapted without degrading the quality of the global model. Second, **GFSTL’s flexible FL+SL aggregation across RSUs and HAPs** reduces the negative impact of user diversity by aligning intermediate representations before global averaging, thus maintaining robustness as user batches scale.

From an ITS perspective, these findings imply that GFSTL is particularly well-suited for large-scale deployments where thousands of heterogeneous edge devices (cars, UAVs, and roadside cameras) contribute data. Unlike FL, SL, or FSL, which deteriorate as the system scales, GFSTL achieves both scalability and accuracy preservation, a property critical for real-time safety and perception tasks in 6G-enabled ITS environments.

### 3.9.4 Latency and Communication Efficiency

Figure 3.9 reports the total end-to-end latency (per training round) as the number of user batches per RSU increases from 2 to 10. The total latency plotted in the figure corresponds to the wall-clock time required to complete a single global update and is computed according to the decomposition introduced in Section 3.6. In that model, the per-round latency can be expressed as the sum of three main components:

$$T_{\text{round}} = \underbrace{T_{\text{comp,clients}}}_{\text{local inference/forward/backward}} + \underbrace{T_{\text{comm,clients} \leftrightarrow \text{RSU}}}_{\text{upload/download}} + \underbrace{T_{\text{agg}}}_{\text{RSU- and HAP-level aggregation}},$$

where  $T_{\text{agg}}$  itself depends on whether aggregation is performed serially or in parallel across groups and whether an additional HAP-level global aggregation step is required (see Section 3.6 for the full derivation).

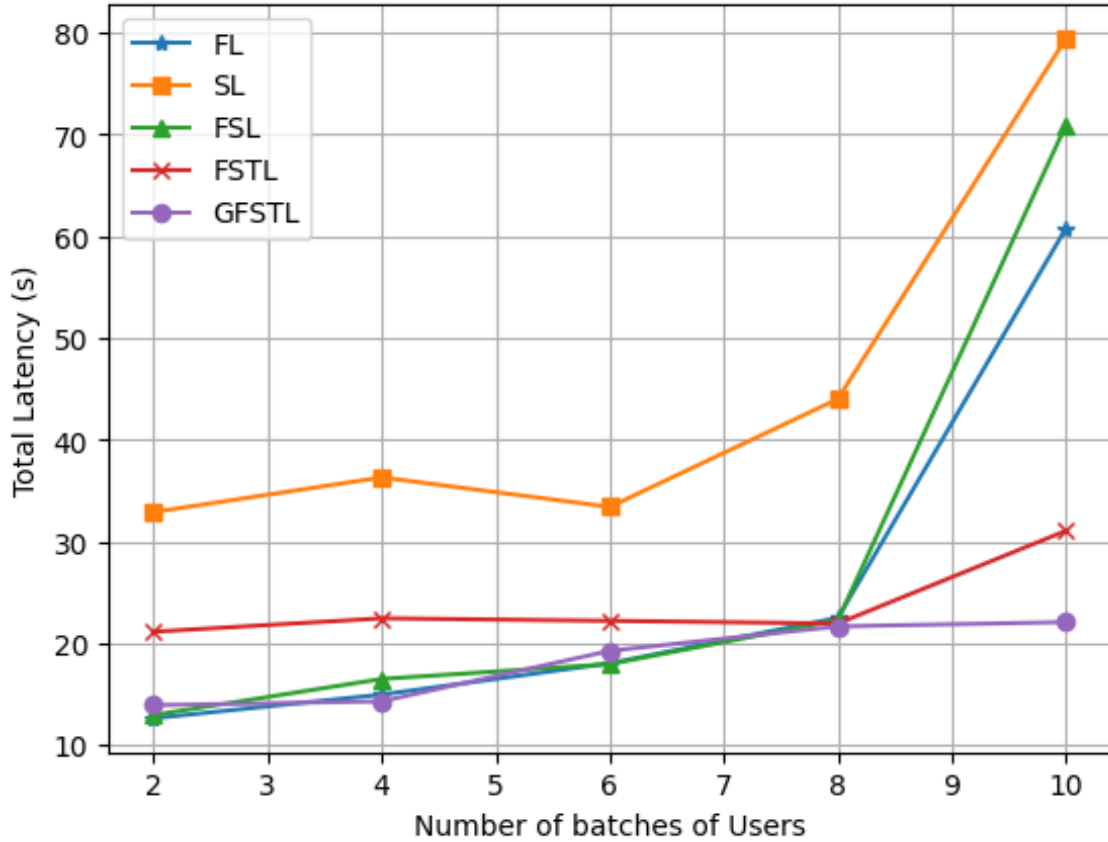


FIGURE 3.9: Latency of different DL methods vs. number of user batches per RSU.

Several important trends are visible in Figure 3.9:

- **GFSTL (purple curve) is effectively flat and lowest at scale.** GFSTL exhibits a nearly constant latency ( $\approx 14$ – $22$  s across the sweep) and shows only a minor increase even at 10 batches. This behavior follows directly from the hierarchical, parallel aggregation used by GFSTL: RSUs aggregate their

connected clients locally in parallel, and only compressed intermediate representations (the smashed tensors) are exchanged upward. In the latency equation, this reduces  $T_{\text{comm,clients} \leftrightarrow \text{RSU}}$  per client and keeps  $T_{\text{agg}}$  dominated by the slowest RSU rather than the sum of all clients, therefore avoiding the explosive growth seen in serial schemes.

- **FSTL (red curve) shows moderate growth but remains substantially lower than FL/FSL/SL at high loads.** FSTL starts around 21 s and rises to  $\approx 31$  s at 10 batches. Compared to GFSTL, FSTL lacks the same level of hierarchical parallelism or dynamic placement of FL/SL servers, so its  $T_{\text{agg}}$  and  $T_{\text{comm}}$  terms increase more with the user count. However, because FSTL still transmits compact representations rather than entire models, it avoids the large spikes observed for full-FL-style approaches.
- **FL and FSL (blue and green curves) remain low on a small scale but spike dramatically at 10 batches.** Both FL and FSL are roughly in the 12–22 s range for small-to-moderate batch counts, then jump to  $\approx 61$  s (FL) and  $\approx 71$  s (FSL) at 10 batches. This non-linear escalation is explained by two mechanisms in the latency model: (i) the upload/download of full model parameters or large gradient vectors causes  $T_{\text{comm,clients} \leftrightarrow \text{RSU}}$  to grow with the number of clients, and (ii) global aggregation in FL requires communication with a centralized server (HAP or cloud) that becomes a bottleneck when many clients simultaneously upload large models. In FSL, the extra split/label-handling complexity can further amplify communication and aggregation overhead, hence the higher spike compared to plain FL.
- **SL (orange curve) is highest overall and grows rapidly at large scale.** SL demonstrates high latency even at small batch counts ( $\approx 33$  s) and reaches  $\approx 80$  s at 10 batches. This is expected because classic Split Learning operates in



a sequential (or partly sequential) manner where server-side processing often waits for client-by-client smashed uploads and sequential forward/backward passes. In terms of latency decomposition, SL's  $T_{\text{agg}}$  effectively becomes a sum of per-client server processing times, causing linear (or worse) scaling with the number of clients.

Putting these observations in the context of Section 3.6, the figure confirms two key messages: (i) reducing the size of exchanged payloads (smashed tensors) drastically lowers  $T_{\text{comm}}$  and thereby amortizes network cost as the user count grows; and (ii) organizing aggregation in parallel groups (RSU-level) and then performing a higher-tier aggregation (HAP-level) bounds the  $T_{\text{agg}}$  term by the slowest group rather than the sum of all client delays, thus achieving much better scalability in wall-clock time.

For latency-sensitive ITS tasks (real-time perception and collision avoidance), the results show that GFSTL and, to a lesser extent, FSTL are preferable because they maintain low per-round latency even as many edge devices participate. Pure FL and SL become impractical at high client density unless additional network capacity or aggressive compression is employed. In summary, Figure 3.9 empirically validates the latency analysis in Section 3.6, demonstrating that compact smashed payloads combined with hierarchical parallel aggregation (the core of GFSTL) deliver superior latency scaling for large-scale multilayer 6G ITS deployments.

### 3.9.5 Summary of Key Results

Table 3.5 presents a compact quantitative comparison of the methods evaluated using the metrics described in previous sections: final detection performance (mAP @ IoU 0.5), rounds required to reach 95% of the final accuracy, end-to-end per-round latency (computation + communication), and per-round communication volume. All

methods were executed using the physical and hyperparameter settings in Table 3.4 so that differences stem from the learning paradigm itself rather than from hardware or network variation. The entries in the table illustrate the trade-offs we discuss in the text: SL achieves low communication but suffers from very high latency due to its serial server–client operation; FL transmits large model updates and therefore incurs the largest communication volume and moderate latency; FSTL and GFSTL reduce both communication and latency by transmitting compact smashed representations and leveraging hierarchical aggregation, with GFSTL delivering the best balance of accuracy (99.8% mAP), fast convergence (two rounds to 95% of the final accuracy), and low per-round latency (160 ms) under the baseline configuration.

TABLE 3.5: Summary of results.

| Method | Final Accuracy | Rounds to Converge | Latency/Round | Comm vol/Round |
|--------|----------------|--------------------|---------------|----------------|
| FL     | 94.2%          | 5                  | 420 ms        | 4.5 MB         |
| SL     | 97.6%          | 10                 | 1500 ms       | 0.9 MB         |
| FSL    | 94.5%          | 8                  | 380 ms        | 0.8 MB         |
| FSTL   | 94.9%          | 3                  | 220 ms        | 0.8 MB         |
| GFSTL  | <b>99.8%</b>   | <b>2</b>           | <b>160 ms</b> | <b>0.6 MB</b>  |

In summary, our evaluation demonstrates that GFSTL, leveraging the YOLOv5 model pre-trained on the Cityscapes Dataset, excels in overcoming the challenges posed by the heterogeneous and dynamic nature of ITS environments. GFSTL consistently achieves higher accuracy and significantly lower latency across various user configurations, affirming its suitability for ITS applications where real-time decision-making and high accuracy are crucial. The GFSTL framework’s capacity to handle large-scale, diverse data sources with minimal communication overhead makes it a powerful solution for intelligent, connected infrastructure within 6G networks.

### 3.10 Discussion

This work presents a unified GFSTL framework designed for 6G ITS, demonstrating how a hierarchical, multilayer network architecture can enhance the scalability, accuracy, and privacy of DL. The proposed architecture effectively combines RSUs and HAPs as FSL and central FL servers, respectively, enabling optimized model aggregation across a diverse set of clients, including VUs, UAVs, and street cameras. This approach directly addresses the need for a cohesive system that can manage the massive scale and data complexity envisioned for 6G ITS.

Our simulation results, which leverage the advanced YOLOv5 model and the realistic Cityscapes dataset, provide strong validation for the proposed framework. The superior performance of GFSTL in terms of faster convergence, higher accuracy, and significantly lower latency compared to traditional DL methods highlights the powerful synergy of combining Federated, Split, and Transfer Learning. The framework's ability to maintain high performance even as the number of users increases underscores its scalability, a critical requirement for real-world ITS deployments. By distributing computational tasks across various network layers and minimizing communication overhead through model splitting, GFSTL is well-suited to the demands of real-time data processing and decision-making in high-mobility environments.

However, it is important to acknowledge the limitations of this study, which can inform future work. Our simulations, while comprehensive, were conducted under certain idealizations, such as uniform data distributions among clients and stable network conditions. Real-world ITS environments will undoubtedly feature significant data heterogeneity from different sensors and perspectives, as well as dynamic

network challenges like intermittent connectivity and bandwidth fluctuations. Furthermore, our latency analysis provides a theoretical foundation, but practical deployments will require accounting for additional overheads related to processing and network management.

Future research should focus on extending the GFSTL framework to address these real-world complexities. A crucial next step would be the development and evaluation of GFSTL in a hardware testbed or a more sophisticated network simulator that models dynamic and unpredictable conditions. Further investigation into advanced aggregation strategies that can effectively handle non-IID data from heterogeneous clients to prevent model drift and ensure fairness is also needed. Finally, exploring dynamic resource management algorithms that can optimize the allocation of computation and communication resources across the multilayer architecture in real time would be a valuable contribution. Building on the foundation provided in this work, future research can further advance the development of an intelligent, interconnected, and resilient transportation ecosystem for the 6G era.

### 3.11 Conclusion

This chapter introduced a unified and intelligent DL framework, GFSTL, designed to meet the complex demands of 6G-enabled ITS operating over multilayer integrated T/NTNs. Our core contribution is a novel architecture that synergistically combines FL, SL, and TL to address the critical challenges of scalability, data privacy, and resource efficiency inherent in large-scale, heterogeneous environments. The proposed GFSTL framework leverages a hierarchical structure, utilizing network elements such as RSUs, HAPs, and LEO satellites as distributed servers to manage and aggregate model training across a diverse range of clients, including VUs, UAVs,

and ground-based cameras. This multilayer approach not only enhances network coverage and resilience but also optimizes computational and communication loads by partitioning model training and minimizing data exchange. Through comprehensive simulations using the advanced YOLOv5 model on the Cityscapes dataset, we have demonstrated the superior performance of GFSTL. Our results validate that the framework achieves significantly faster convergence, higher model accuracy, and lower end-to-end latency compared to traditional DL methods like FL and SL, particularly as the number of users and data complexity increase. By successfully integrating advanced learning paradigms within a scalable T/NTN architecture, this work provides a robust and privacy-preserving solution for next-generation ITS. The GFSTL framework lays a foundational stone for the development of truly intelligent, connected, and resilient transportation ecosystems, paving the way for the ambitious vision of 6G to become a reality.

## Chapter 4

# Implementation and Performance Analysis of Distributed Learning Frameworks

Some content of this chapter is based on the following articles [44, 109]:

1) **David Naseh**, Mahdi Abdollahpour, and Daniele Tarchi. "Real-world implementation and performance analysis of distributed learning frameworks for 6G IoT applications." *Information* 15, no. 4 (2024): 190.

2) Lorenzo Ridolfi, **David Naseh**, Swapnil Sadashiv Shinde, and Daniele Tarchi. "Implementation and evaluation of a federated learning framework on Raspberry Pi platforms for IoT 6G applications." *Future Internet* 15, no. 11 (2023): 358.

## 4.1 Introduction

This chapter lays the groundwork for understanding how Distributed Learning (DL) can be practically realized and evaluated in the emerging 6G-enabled IoT and IoV landscape. As billions of heterogeneous devices come online, centralized model training becomes increasingly impractical due to bandwidth constraints, privacy concerns, and device resource limitations. To address these challenges, this chapter first investigates a fully operational FL framework built with commodity Raspberry Pi boards and Virtual Machines (VMs), orchestrated via Flower and PyTorch. By tackling real-world variables such as device heterogeneity, data imbalance, computational throttling and thermal throttling, and by leveraging techniques like pre-training and Knowledge Transfer (KT), this work exposes the interplay between onboard resources, communication costs, and model accuracy in a 6G setting.

Building on these insights, the chapter then transitions to a comparative study of conventional FL versus FTL across a mix of edge platforms—including Raspberry Pi, Odroid, and cloud VMs. Through extensive experiments measuring accuracy, convergence speed, resource utilization (CPU load, memory, temperature), and energy consumption, it demonstrates that FTL not only accelerates training and boosts accuracy but also curbs power draw and remains robust under client dropout conditions. To summarize, this chapter furnish a holistic framework—spanning from low-cost hardware testbeds to optimized TL strategies—for deploying scalable, privacy-preserving distributed intelligence in tomorrow’s fully connected IoT ecosystems.

## 4.2 Implementation and Evaluation of a Federated Learning Framework on Raspberry PI Platforms

The upcoming 6G technology is expected to create an intelligent, fully connected, and digitized society through large-scale deployments of distributed intelligent networks [116]. This is expected to create a plethora of new intelligent services and applications with specific demands. Internet of Things (IoT) technology has foreseen great success through enabling 5G solutions and is expected to play a key role in 6G society [117]. With the integration of IoT subcases into different wireless scenarios, many sensory nodes are expected to be deployed in the 6G world, with the ability to sense nearby environments and produce tons of high-quality data. This data can be extremely important to enable intelligent solutions in the 6G networks [118].

Machine Learning (ML) is another important technology that has acquired a central role in the 6G vision; in particular, to enable intelligent services [119, 120]. Various ML methods are expected to be deployed in different wireless scenarios to enable intelligent solutions. ML can be extremely useful for analyzing the 6G network data and harnessing its intelligence. The Centralized Learning (CL) method is one of the most common approaches to generating AI algorithms in distributed communication scenarios. This involves concentrating all training data on a single server, which can complete the model's training individually due to its high computing power. However, this solution may not be particularly efficient when analyzed from a 6G perspective. Next-generation networks will need to handle large amounts of information, most of which will be produced by devices at the network's edge. In this scenario, moving data to a central server is expensive in terms of communication resources and represents a potential risk of network overload. It can induce large



communication overheads and has limited applicability in the resource-constrained and time-critical 6G world. Furthermore, privacy and security restrictions can also limit the possibility of data transmissions to server nodes.

A more efficient alternative is offered by DL, which is based on decentralization of training [120]. In this approach, the training phase will be carried out directly on end devices or possibly on specific nodes located strategically on the edge of the network. DL allows for the training of models by processing the local data of each device directly within it. This approach enables the use of large amounts of heterogeneous data scattered throughout the vast network, involving a larger audience of users than the centralized variant. This approach also introduces additional advantages. First, it makes it possible to significantly reduce data flow, communication overhead, and network traffic. Second, there is greater protection for end-user privacy as data are processed directly on personal terminals. Finally, by distributing the computational load among the multitude of devices, it is also possible to obtain essential improvements in terms of energy management. Several forms of DL, such as Federated Learning (FL), collaborative learning, Split Learning (SL), and Multi-agent Reinforcement Learning (MARL), are widely considered in wireless networks for enabling intelligence-at-the-edge. Among others, FL has achieved great success in building high-quality ML models based on dispersed wireless data [121, 122]. FL is also a candidate for next-generation 6G communication standard allowing for setting up an intelligence-at-the-edge framework [123].

The traditional FL approach includes a set of devices with datasets and a server node [121]. The server node initiates the learning process by defining the learning task, and corresponding model, which is then transmitted to FL devices. With the help of received model parameters from the server node (also called the global model), their datasets, and onboard computation capabilities, each FL device is

expected to train the ML model locally. Then, the updates to the ML model from each device are sent to the server. After receiving the updates from the devices, the server node can use the aggregation function (e.g., Federated Averaging (FedAvg)) to create a new global model. This aggregation phase allows devices to share their training knowledge. The devices then use the new global model in the next round of model local training. The process lasts for several rounds till some pre-defined stopping criteria are fulfilled. Though this approach has several advantages in terms of reduced data transmission costs, enhanced data privacy, etc., and several new challenges have emerged. One of the major challenges in the FL framework is the presence of heterogeneous nodes with different capabilities, in terms of available datasets, computation power, etc. The presence of heterogeneous nodes can be common in different 6G scenarios. Also, the amount of time required to achieve the convergence of the FL model can be unacceptable in latency-critical 6G use cases. Therefore, it is important to analyze the performance of the traditional FL approach in the presence of heterogeneous devices to have a common understanding of their behaviors and possible solutions to tackle the challenges.

### 4.2.1 Technological Background

ML technology has gained a lot of attention for enabling intelligent solutions in wireless networks including mobile communication networks [124], wireless sensor networks [125], transportation systems [121], Non-Terrestrial Networks (NTNs) [126]. Complex wireless communication problems such as resource management [127], data offloading toward edge networks [128], spectrum management [129], routing [130], user-server allocation [122], etc., can effectively solve through different ML techniques. Among others, FL is widely considered a DL approach to provide efficient

learning solutions. In one paper [131], the authors proposed energy-efficient FL solutions in wireless communication environments. Work in [132] discusses the applicability of FL solutions in smart city environments. FL is also widely used to solve Vehicular Network (VN) problems, especially in Edge Computing (EC) environments [121]. FL solutions are also considered over different satellite networks [133]. However, these works have analyzed the performance of FL solutions in different wireless environments, without considering the practical implementations.

Recently, several authors considered a learning testbed implemented through different sensory nodes to measure the ML solution's performances. Raspberry Pi devices are commonly considered to analyze the performance of ML solutions proposed to solve different wireless networking problems. In [134], the authors proposed Reinforcement Learning (RL)-based solutions for efficient routing processes in software-defined wireless sensor networks. The testbed includes Raspberry Pi devices as sensor nodes to analyze the performance of RL solutions. In one paper [135], the proposed deep learning-based solutions for detecting a speed bump in an intelligent vehicular system to assist drivers are tested with the help of Raspberry Pi devices and associated camera modules. In one paper [136], an approach called Distributed Incremental Learning (DIL) was introduced to mitigate "Catastrophic Forgetting" in healthcare monitoring. However, the large model size (49 KB) poses challenges for Raspberry devices, and there is a lack of data batch quality details affecting convergence.

In one paper [137], the proposed communication-efficient FL solutions are tested with the help of Raspberry Pi devices in EC environments. However, to the best of our knowledge, the most recent literature lacks the study associated with the analysis of FL performance in the presence of heterogeneous clients. Given the importance of different IoT subsystems with heterogeneous nodes, such studies can

be extremely useful from the 6G network’s perspectives. This is one of the main motivations behind this experimental analysis of the FL process in the presence of diverse sets of clients.

## 4.2.2 Contributions and Novelties

In this study, we present an in-depth exploration of FL methodology within the context of 6G technology, delving into its intricacies and challenges across heterogeneous nodes. Our work uniquely integrates hardware and software components, utilizing a distinctive combination of Raspberry Pi devices and VMs as FL clients, each equipped with diverse datasets sourced from the CIFAR10 dataset—a widely acknowledged benchmark in image classification. Our contributions and innovations can be outlined as follows:

- **Cooling Mechanism Impact (Section 4.2.6.1):** We meticulously investigate the influence of cooling mechanisms on training accuracy, underscoring their practical significance in accelerating model convergence, especially in resource-constrained environments. This detailed analysis, expounded on in Section 4.2.6.1, elucidates the pivotal role of cooling mechanisms, providing valuable insights into optimizing FL performance.
- **Heterogeneous Client Compensation (Section 4.2.6.2):** Through a thorough exploration of asymmetric data distribution scenarios, both with and without random selection, we dissect the intricate dynamics of FL performance. Our study highlights the delicate balance necessary in distributing training data among diverse nodes, revealing the complexities of FL dynamics

in real-world scenarios. These findings, presented in Section 4.2.6.2, offer critical insights into the challenges and solutions concerning data heterogeneity in FL setups.

- **Overfitting Mitigation Strategies (Section 4.2.6.2):** We tackle the challenge of overfitting in FL by implementing meticulous strategies. By integrating random selection techniques, we effectively mitigate overfitting risks, optimizing model generalization and ensuring the resilience of FL outcomes. This contribution, outlined in Section 4.2.6.2, underscores our commitment to enhancing the robustness of FL models.
- **Scalability Analysis (Section 4.2.6.3):** Our study provides a comprehensive exploration of FL scalability, assessing its performance with an increasing number of users. This analysis, detailed in Section 4.2.6.3, offers crucial insights into FL's scalability potential, essential for its integration in large-scale, dynamic environments. It emphasizes the system's adaptability to diverse user configurations, laying the foundation for FL's applicability in real-world scenarios.
- **Pretraining Effectiveness (Section 4.2.6.4):** We delve into the effectiveness of pretraining techniques in enhancing accuracy rates. Pretraining emerges as a potent tool, significantly boosting the model's performance and showcasing its potential in optimizing FL outcomes. This contribution, discussed in Section 4.2.6.4, highlights the practical implications of pretraining in FL applications, providing actionable insights for future implementations.
- **Transfer Learning Impact (Section 4.2.6.5):** In Section 4.2.6.5, we investigate the potential of Transfer Learning (TL), evaluating its impact under diverse client configurations. Our results underscore TL's capacity to enhance

FL model performance, especially in the face of varied client scenarios [57]. This analysis showcases TL’s adaptability in real-world applications, emphasizing its role in improving FL outcomes across dynamic and heterogeneous environments.

These contributions collectively form a comprehensive and innovative exploration of FL dynamics, addressing key challenges and offering practical solutions essential for the advancement of FL technologies in complex, real-world settings.

### **4.2.3 Limitations**

Despite these contributions, our study acknowledges certain limitations. While our findings offer valuable insights, the scope of this research is confined to specific FL configurations and dataset characteristics. Further exploration of different FL architectures, diverse datasets, and real-world deployment challenges remains an area ripe for future investigation. Additionally, the scalability analysis, while comprehensive, focuses on a limited range of users and could benefit from further exploration with more extensive user groups in practical scenarios.

### **4.2.4 Distributed Machine Learning**

The traditional ML approach was based on the centralized structure, where distributed wireless nodes, a potential data source, were needed to transmit their data samples to the centralized, more powerful node with a large amount of storage and computation power. With growing interest in the 5G system and the upcoming 6G technology, the wireless world is filled with tiny devices capable of sensing the environment and generating tons of high-quality data. Such data can be extremely

large, and if a traditional centralized approach is adopted, it can induce a huge communication overhead. On the other hand, with the presence of such a large number of devices, the global dataset generated at the centralized server node (i.e., through the accumulation of data from the distributed nodes) can be extremely large, inducing much higher training costs. In addition to this, novel intelligent services and applications are based upon stringent requirements in terms of latency, privacy, reliability, etc. This presents challenges when considering centralized ML model training for wireless scenarios. However, with recent innovations in hardware/software domains, the end devices' onboard capabilities have increased by several folds. With this new capability, these devices can train fairly complex ML models locally with their own datasets. This can omit the requirements for long-distance data communication and additional training overhead. In addition to this, devices can communicate with each other and server nodes to fine-tune the ML models with improved performances. This has opened a new trend of ML model training called DL, for countering the drawbacks of traditional centralized methods. There are various forms of DL methods considered in the recent past.

There are two main approaches available for performing DL: with data in parallel or with models in parallel [138]. The former involves distributing training data on several servers, while the latter divides the model's parameters between different servers. However, implementing the parallel model approach is difficult due to the complexity of dividing ML models into distinct groups of parameters. Therefore, most distributed ML implementations work through data distribution. FL, collaborative learning, MARL, and SL are some of the most important DL methods. Among others, FL has been widely used in wireless networks to enable intelligent solutions efficiently.

FL is a framework for distributed ML that protects privacy by operating through

decentralized learning directly on end devices. It involves a certain number of clients, distributed throughout the network, each of whom trains their local model using the data at their disposal. After training, the clients send their models to a central server, which aggregates them into a single Neural Network (NN) and then transmits it to the end devices. This is an iterative process, with each iteration called a federated round. The objective of FL is to minimize a function and ensure efficient FL, several variables must be considered [139].

For the efficient implementation of FL, it is imperative to take into account several variables, encompassing the following:

- Selection of devices for learning
- Disparities in performance levels among the clients in use
- Management of heterogeneous training data
- Potential algorithms for local models' aggregation
- Selection of a proper aggregation Strategy at the Parameter Server
- Resource allocation

Concerning the choice of devices, specific parameters, including the quality and quantity of local data, connection performance with the central server, and computational performance of the client, need to be evaluated.

In a heterogeneous environment that involves wireless nodes with various onboard capabilities, it is crucial to pay attention to differences in the computing capabilities of the devices. In fact, a client with reduced computational capabilities will require a longer time to train the model locally, thereby risking the deceleration of the entire learning process. The following two FL approaches are widely considered to enable



DL [140] and can be impacted by the heterogeneous nature of the computational capabilities of the devices:

- **Synchronous FL:** All devices participate in training the local models for a specific period, sending the parameters to the central server. In this case, the server receives the client models simultaneously and aggregates them with the certainty that it is using the contribution of all the devices. However, this approach poses some challenges, in the case of heterogeneous client nodes having different capabilities. In such cases, the less-performing clients are compelled to invest more resources to complete the training within the expected timeline. To match the latency performance of other, high-performing clients with more resources, devices can only use a subset of their data.
- **Asynchronous FL:** In this case, there are no time restrictions for local training operations, with each device training its model based on its own capabilities, after which it sends the parameters to the server that proceeds with aggregation. This approach is more appropriate even in the presence of unstable network connections, where a device without network access can continue to train its model until it reconnects. Such an asynchronous approach can potentially reduce the number of FL devices participating in the individual FL rounds. This also requires more complex server-side operations to manage the devices according to their needs.

In Section 4.2.6, we explore these FL approaches when evaluating the system performance; more specifically, we employ asynchronous FL in heterogeneous client compensation (Section 4.2.6.2), to tackle the challenge posed by the discrepancy between the relatively sluggish Raspberry PIs and the swift VMs, which ultimately

leads to a decline in the overall accuracy of the aggregated data. This measure shall be taken to restore the balance between the two.

In another case, the presence of heterogeneous amounts of data on FL devices can also largely impact FL performance. In one paper [141], the authors propose federated continual learning to improve the performance of Non-IID data by introducing the knowledge of the other local models; however, the paper does not address the scalability of the proposed method for large-scale DL systems. In such scenarios, locally generated models at different FL nodes may have distinct characteristics from each other and may not lead to proper convergence when aggregated into a single NN. The convergence problem can directly impact the algorithm chosen for the aggregation. In particular, the traditional federated average (FedAvg) approach can have limited performance since it does not distinguish the model parameters from different devices (i.e., simple averaging) [140]. In another case, proper weights can be assigned to each device's models according to their quality (i.e., weighted average). In some cases, device selection policies can be adapted to avoid the participation of FL devices with imperfect models.

The use cases mentioned above indicate the importance of defining a proper FL framework in heterogeneous environments based on the properties of the FL device and the characteristics of the environment. A one-fit-all FL approach can have reduced performance in different cases. Therefore, it is vital to analyze the performance of FL models in various scenarios and to select a proper FL model.

### 4.2.5 Implementation of the System

The primary goal of this work is to design and analyze a comprehensive and well-structured FL framework to train ML algorithms using a federated approach involving heterogeneous FL clients. Here, we outline the important steps considered during the implementation of the FL process.

The considered FL system is based on a client-server architecture, with the server represented by a Windows PC and a set of Raspberry Pi devices acting as FL clients. In addition, a set of virtual clients are also considered, to build an FL framework with heterogeneous clients. Given the importance of network programmability and virtualization technologies in the 5G/6G networks, defining the FL framework with a set of hardware/software clients can have added advantages. Finally, inter-device communication is carried out through the local network, to which all hardware nodes are connected via Wi-Fi.

As mentioned above, the main objective is to evaluate the efficacy of FL in the context of devices with limited and heterogeneous resources.

We have considered a typical image classification problem and aim to build a proper DNN with the help of the considered FL framework. The well-known CIFAR10 dataset is considered during model training operations. It should be noted that to induce data heterogeneity over different FL clients, the original dataset is split into different datasets. Also, though the analysis is performed with a specific ML task with predefined datasets, this can be extended to any generic ML problem. The FL framework is built with the help of Python programming language. In particular, the PyTorch library is considered to train the NN model, while Flower, a specialized library for federated ML, is considered to automate the client-server interactions more efficiently [142].

In the following, we describe in detail the various configurations used in client/server parts of the considered FL model. Figure 4.1, presents the basic elements of the considered FL framework that includes a set of Raspberry Pi devices, VMs, and an FL server.

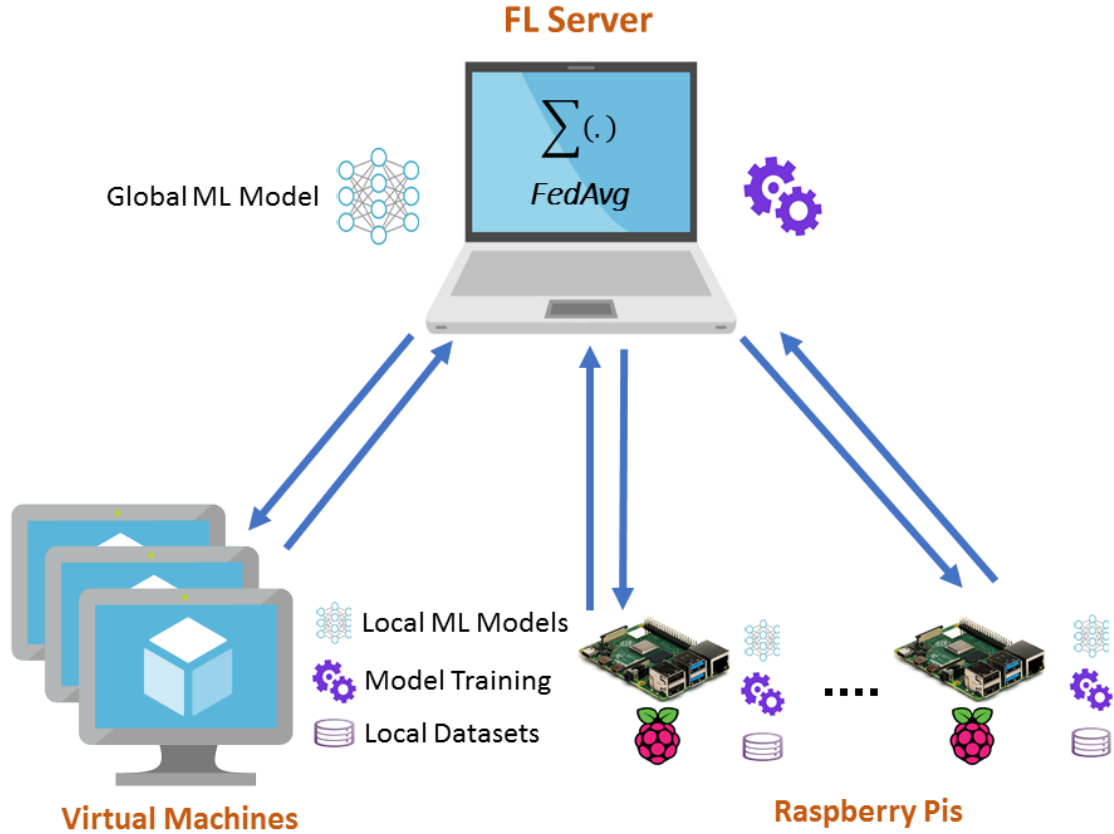


FIGURE 4.1: Considered FL framework with heterogeneous clients.

#### 4.2.5.1 Server Configurations and Functionalities

Here we introduce the main steps required to configure the server part of the FL system and the corresponding software employed. The FL server is installed on a Windows computer, which exhibits high-performance levels compared to the client devices. Specifically, the server is an Asus ROG Zephyrus S GX502GV with an Intel Core i7-9750H 6 x 2.6–4.5 GHz, Coffee Lake-H processor, NVIDIA GeForce

RTX 2060 Mobile (6 GB VRAM, GDDR6 graphics card) and 16 GB DDR4–2666Hz RAM. Furthermore, Wi-Fi connectivity is 802.11 a/b/g/n/ac with Intel Wireless-AC 9560.

We used the Anaconda platform, which allows the development of Python-based solutions with advanced package and library management. The current project was developed using the basic Anaconda environment, which includes Python 3.9.13. We downloaded and installed two further libraries, namely PyTorch version 2.0.0 and Flower version 1.3, through the integrated terminal. The framework is built around three Python scripts: one to run the server, another to run the client, and a third to define specific NN training methods. It should be noted that the system can only communicate on the local network. Data are transmitted and received between the various devices using specific methods defined in the scripts.

The FL server needs to perform several functions to enable the FL. At first, the server establishes communication with a considered set of clients through the local network. Next, it initializes the global DNN model. DNN model can be initialized through random parameters or pre-trained models saved in memory based on the considered scenarios. After that, the network parameters are transmitted to all clients to have a common starting point for federated training.

Throughout the federated training process, clients train their models locally for a specified number of local rounds. During each round, each client computes a local model update using its own data and sends the respective model parameters to the server. The server then aggregates the local updates from all clients into a single global model. This aggregation is performed using the FedAvg algorithm, which ensures that the global model is updated based on the weighted average of all local updates. The process of computing local model updates, sending them to the server, and aggregating the updates into a global model continues iteratively until

convergence. FedAvg enables collaborative model training across multiple devices or clients while preserving data privacy. The detailed procedure and formula for the FedAvg algorithm are outlined in Algorithm 4.1.

---

**Algorithm 4.1** FedAvg Algorithm

---

**Input:** Global model parameters  $\theta$

**Output:** Updated global model parameters  $\theta$

```
1: Initialize  $\theta$  with random values
2: while not converged do
3:   for each client  $i$  in the federated network do
4:     Compute local model update  $w_i$  using client's local data
5:     Send  $w_i$  to the server
6:   end for
7:   Aggregation (FedAvg):
8:     Compute updated global model parameters  $\theta$  as the average of received
       local updates
9:      $\theta \leftarrow \frac{1}{N} \sum_{i=1}^N w_i$ 
10: end while
```

---

The performance of the aggregated model is then assessed in terms of accuracy and losses using a pre-loaded test data set on the server. The results of this assessment are then recorded in a CSV file. After that, the new aggregated model is returned to the clients to repeat the training procedure with the next federated round. The described process is reiterated until the designated number of iterations is reached. Each of these functions is important for enabling the FL process.

#### 4.2.5.2 Client Configurations and Functionalities

Client devices include a set of Raspberry Pi devices along with the virtualized clients in the form of VMs. In particular, Raspberry Pi 3B+ is considered as a client during the experiments. The Raspberry Pi 3B+ is equipped with a Broadcom BCM2837B0 processor, Cortex-A53 (ARMv8) 64-bit SoC with a clock speed of 1.4 GHz, a 1GB LPDDR2 SDRAM, and 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless LAN. Moreover, it has Bluetooth 4.2, BLE, Gigabit Ethernet over USB 2.0 (max throughput 300 Mbps), a 40-pin GPIO header, Full-size HDMI, 4 x USB 2.0, DSI, a Micro SD slot, and a power supply of 5V/2.5A DC.

To facilitate the experiment, the Raspberry PI 64-bit Operating System (OS) was installed on each Raspberries through the official imager. The 64-bit system is necessary for the proper functioning of PyTorch, which currently does not support 32-bit variants. Configuring the devices via SSH enabled the Virtual Network Computing (VNC) service, which uses the local network to transmit the Raspberry desktop to the connected Windows computer. The remote connection via the cloud could further extend the functionality of VNC. Such a procedure allowed interaction with the Raspberry desktop directly on the notebook that hosts the server, facilitating monitoring of the simulation's progress from a single screen.

The client-side code was executed in the Python environment pre-installed on the Raspberry Pis and updated to version 3.9.2. Moreover, the installation of the PyTorch and Flower libraries was necessary. The clients are based on the `client.py` script, which inherits all the contents of the file `cifar.py`. This file is essential for generating, training, and evaluating the NN. Such a configuration allowed the devices to participate in the training process by sharing their computational power with the server. In this way, training time and computational costs can be reduced.

The Raspberry Pi 3B+ devices were a suitable choice for this experiment due to their low cost and high customizability. They allow users to modify the hardware and software configurations, facilitating the implementation of ML models and algorithms. Raspberry Pi devices can also be used in various applications, such as robotics, automation, and the IoT. The proposed setup could be replicated in various scenarios where training ML models on devices with limited resources could be beneficial.

In addition to the Raspberry devices, up to eight virtual clients are also considered during the experimentation. It should be noted that all the virtual clients were installed on the same PC. However, this can be easily extended to multi-PC scenarios to enable more diverse sets of clients. Such an approach is beyond the scope of this work. In general, virtual clients have more resources compared to hardware clients. The virtual clients are based on the same scripts used in the implementation of the physical clients, and their execution simultaneously is facilitated using multiple Python terminals open on different screens. A cooling fan was necessary to prevent excessive degradation of computational performance in the Raspberry devices due to overheating. The fan was operational during all the tests conducted to maintain the optimal performance of the devices, as indicated in Figure 4.2. Later, in the simulation and performance evaluation section, we explore the effect of cooling on accuracy and convergence rate.





FIGURE 4.2: Cooling mechanism for Raspberry Pi devices.

In the FL framework, client nodes are required to perform a distinctive set of functions. In the beginning, each client should establish a communication channel with the server through a local network. At the beginning of each FL round, the client should receive the updated global model parameters from the server machine. These parameters along with the local data should be used to update the ML model through local training operations. Client devices should train the NN on local data, iterating for a certain number of periods, i.e., local epochs. Furthermore, it is important to evaluate the performance of the newly trained local model using a local test data set and send the obtained metrics to the server. Finally, the client should transmit the parameters of the trained model back to the server. Figure 4.3, provides a detailed experimental setup used during the implementation of FL with the help of Raspberry Pi nodes, a virtual machine, and an FL server installed on a Windows PC.

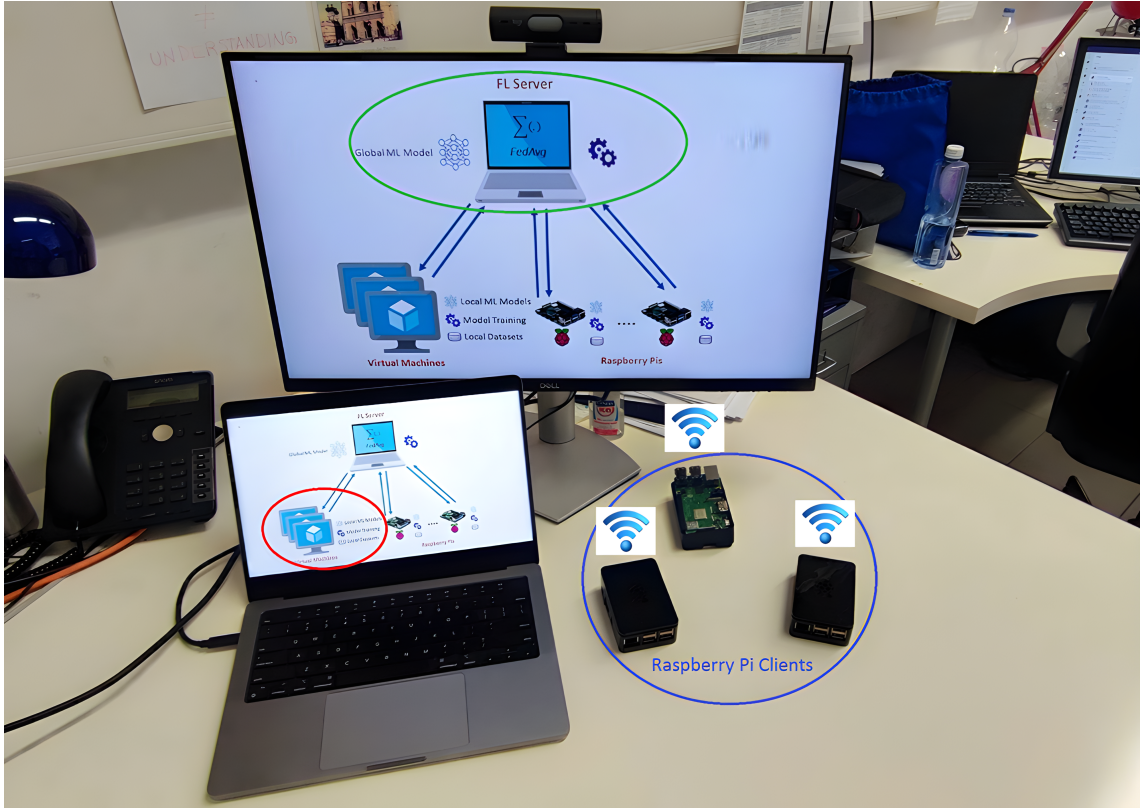


FIGURE 4.3: Experimental setup used during the FL implementation.

#### 4.2.6 Simulations and Performance Evaluations

In this section, with the help of the Python simulation environment, we analyze the performance of the FL framework proposed before. For a considered image classification problem, the CIFAR10 dataset is used. It contains 60,000 colorful pictures, with a resolution of  $32 \times 32$  pixels, separated into 10 classifications with 6000 pictures in each category. The CIFAR10 incorporates 50,000 samples for training and the remaining 10,000 samples for verification. The images are randomly arranged while maintaining a perfectly uniform distribution of the classes. In the training set, each client has precisely 5000 images. In the experimental setup considered, two Raspberry Pi devices are considered as well as up to 8 virtual clients. In our setup, each virtual client was allocated an equal share of CPU resources on the 8-core host

machine. This means that the CPU resources were evenly distributed among the virtual clients, ensuring a fair and consistent experimental environment. By allocating an equal portion of the available 8-core CPU to each virtual client, we maintained a balanced and representative simulation, allowing us to assess the FL framework's performance accurately. This approach ensures that the results obtained were not influenced by uneven resource distribution among the virtual clients, providing a reliable basis for our experimental findings. At first, the CIFAR10 training set was distributed among the clients. The 10 FL iterations are considered with 3 local training epochs. The server uses the test data of 10,000 CIFAR10 samples throughout the simulation. The accuracy of the model was determined using a metric that calculated the percentage of correct predictions among all predictions. The accuracy values presented in our figures are indeed normalized, ranging from 0 to 1, and not represented in percentages. For example, an accuracy value of 0.7 corresponds to 70 percent. Note that, to avoid the extensive training process we have limited the number of training iterations and the overall data size. However, this also upper-bounds the overall performance of DNN. In the experimental studies considered, DNN can only achieve an accuracy of up to 65%. However, the performance can be improved with additional resources, i.e., data samples, devices, training iterations, etc.

Figure 4.4, presents the performance of the FL framework considered in the basic settings, that is, a set of users having the same amount of data and onboard capabilities. This simulation is limited to Raspberry PI devices only. Both centralized and FL models are trained for 30 epochs. To have an adequate comparison over different training epochs, both models' performance in terms of accuracy is plotted with respect to the incremental values of the training epoch. It should be noted that in a considered simulation, the overall epochs represent the overall training process iterations, and therefore for the case of FL, the epochs are based on the number

of FL iterations (rounds) times the local epoch performed. From Figure 4.4, it can be visualized that the FL framework can emulate the performance of a centralized approach in close proximity. Though FL can have a slightly reduced performance compared to the centralized case, the added advantage in terms of reduced communication overheads, and enhanced data privacy can be crucial advantages in the novel wireless scenarios. It also highlights that all the components of the proposed FL framework are configured properly and thus the platform is ready to perform the additional experimental steps.

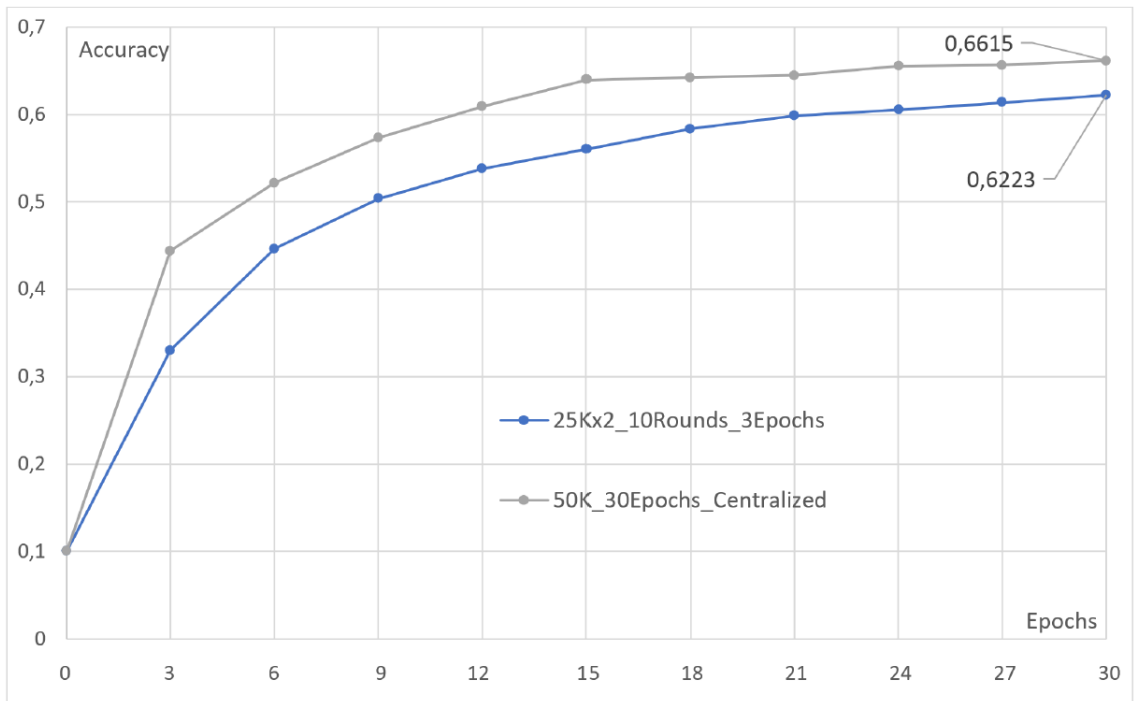


FIGURE 4.4: Accuracy of FL for 2 clients compared to the centralized benchmark vs. epochs.

In the next parts, to evaluate the performance of the models trained with FL, they are compared with a CL benchmark with the same NN. Some of the key variables considered during the experimentation are the number of clients participating in the simulation, the amount of data used for learning, and the selection and partitioning of the training data. Simulations are conducted to assess the impact of a single

variable as well as different combinations. Additionally, performance differences between virtual clients implemented on Windows computers and hardware clients of Raspberry PI are carefully analyzed. The introduction of pre-trained models and alternative learning paradigms such as TL are also considered.

#### **4.2.6.1 Effect of a Cooling Mechanism**

The excessive amount of heat generated by the computation hardware can have a severe impact on the environment and is underlined quite often. Such issues can also affect the performance of the device, which can impact the model training performance. For the case of FL, it is important to analyze the impact of such heating issues on the training performances given the involvement of a large number of sensory devices. Therefore, in Figure 4.5, we have presented the performance of two FL models with similar tasks and training environments. One of the FL studies involves the utilization of a cooling fan to reduce the heating issues of Raspberry Pi clients. As is evident from this figure, accuracy can be improved with the use of cooling devices, which can help to achieve model convergence rapidly. This highlights the importance of incorporating novel cooling mechanisms into end devices to enable efficient intelligent solutions in wireless networks. As such, all subsequent simulations were conducted in the presence of a cooling fan.

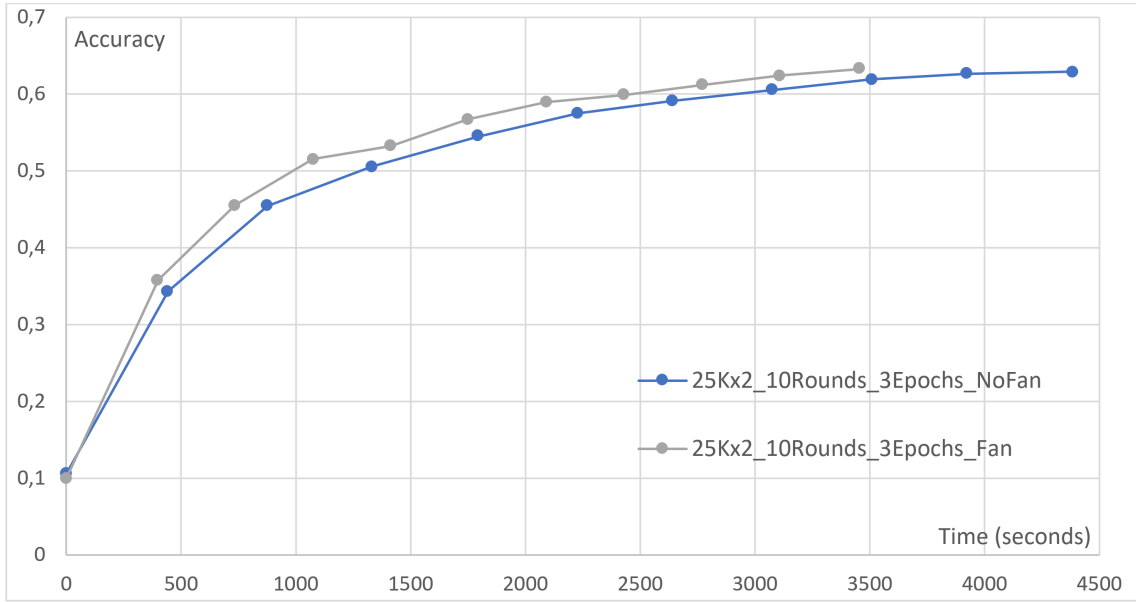


FIGURE 4.5: The effect of a cooling fan on the accuracy of training.

#### 4.2.6.2 Heterogeneous Client Compensation

One of the significant challenges in implementing DL techniques such as FL is the presence of heterogeneous client devices with different amounts of resources, i.e., computational capabilities, training data, etc. In a considered FL implementation two sets of clients are considered. While implementing the FL solutions, a significant disparity in performance was promptly observed between clients operating as VMs and those operating on Raspberry Pis.

In the case of the traditional FL approach with heterogeneous sets of clients, the server node waits until it receives the updates from all the clients in question. In a considered FL setup, Specifically, it has been observed that the server waits until even the Raspberry Pi clients complete their training, even after the most powerful clients have finished their training. We have implemented two approaches to mitigate this waiting time and make the best use of the highest-performing clients. The first approach involves training the best devices for several rounds before transmitting



the model to the server. In contrast, the second approach involves training all clients with the same number of periods but using more data for high-performance clients. Raspberry Pi devices with their limited capabilities often take longer duration to communicate their model updates adding a large amount of communication overhead. On the other hand, virtual clients with a significant amount of resources are able to conclude the learning process promptly, they suffer due to the poor behavior of hardware clients. To counter this issue, we have adopted two different strategies. In the first case, we have normalized the FL iteration time by inducing the harsh local training conditions over the virtual machine-based clients by increasing the number of local epochs performed. With this approach, instead of staying in an idle state and waiting for the parameter updates from the slow-performing clients, the virtual clients try to optimize the local model performance through more training. In the second approach, we have normalized the FL iteration time through different data splits. In this case, each node performs the same amount of training epoch; however, the number of data samples considered at a virtual machine is significantly higher than the Raspberry Pi clients.

The FL data split can be performed with different methods. In Figure 4.6, we have presented the FL model performance with asymmetric data split between heterogeneous clients. Each subfigure includes a centralized benchmark with 50 K samples, an FL approach with a 4:1 split, and another FL case with a 3:2 split. The subfigure on the left is based on the deterministic data split approach, where the repetition of data samples at different nodes is avoided. While in the subfigure on the right, a random data selection approach is adopted, without taking into account the repetitions of data samples at different devices. There is a significant gap between the performance of deterministic and random selection approaches. The deterministic approach can improve the accuracy of the FL model by up to 3.3% compared to the

random case.

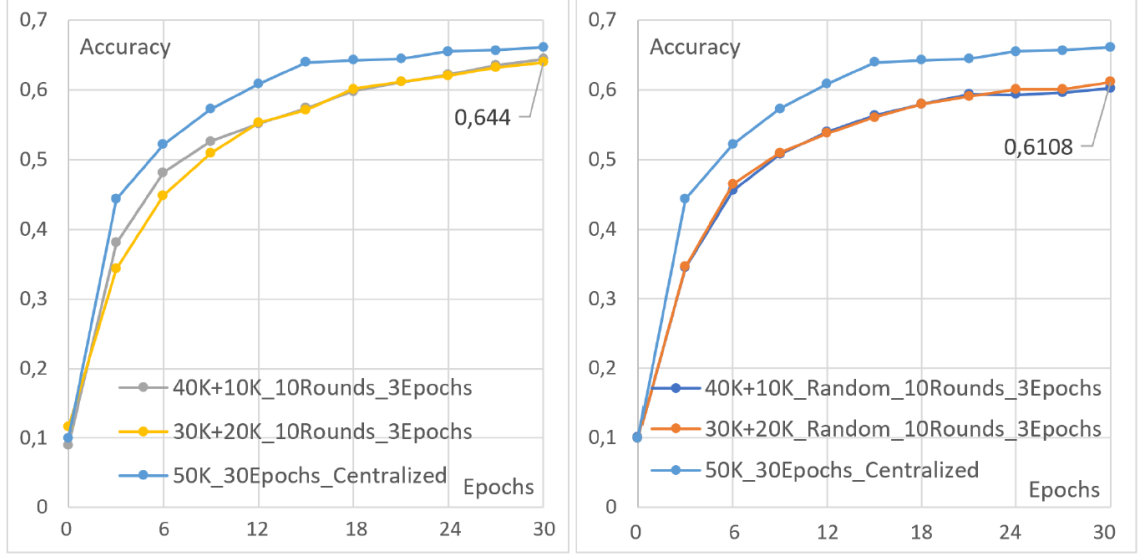


FIGURE 4.6: Simulations with asymmetric data distribution, without and with random selection compared to the Centralized Benchmark.

In the next case, we varied the local training epoch over different clients to analyze the performance. In this case, all devices use the same amount of training samples, while varying the nature of the local training process. In particular virtual machine-based clients perform more local epochs compared to the hardware nodes, before communicating their updates to the parameter server. Figure 4.7 compares the accuracy of different data split options along with the differing local training processes. In this case, asynchronous data split achieves higher accuracy, compared to the two FL cases where the same amount of data is used by the FL clients while normalizing the FL process time through the adaptive local training operations. To verify this trend and lend it further credence, it was deemed necessary to perform several similar tests. These tests were carefully arranged so that their execution times were identical, thus rendering it feasible to obtain a more precise comparison in the time domain. This is demonstrated in Figure 4.8, which compares



the two aforementioned methods for distributing the workload between two heterogeneous clients through several simulations. These simulations entail the following conditions:

- Two clients, each with 25,000 images. The first client undergoes two local epochs, while the second undergoes eight.
- Two clients, each with five local epochs. The first client is assigned 10,000 images, while the second is assigned 40,000.
- Two clients, each with 25,000 images. The first client undergoes three local epochs, while the second undergoes twelve.
- Two clients, each with seven local epochs. The first client is assigned 10,000 images, while the second is assigned 40,000.

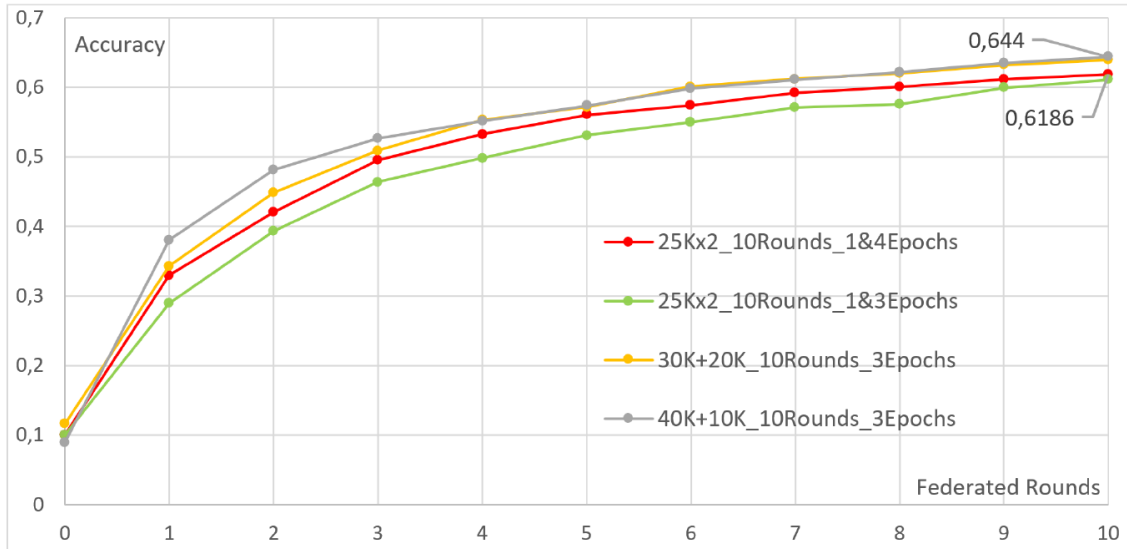


FIGURE 4.7: Accuracy with asymmetric distribution of data vs different numbers of local epochs.

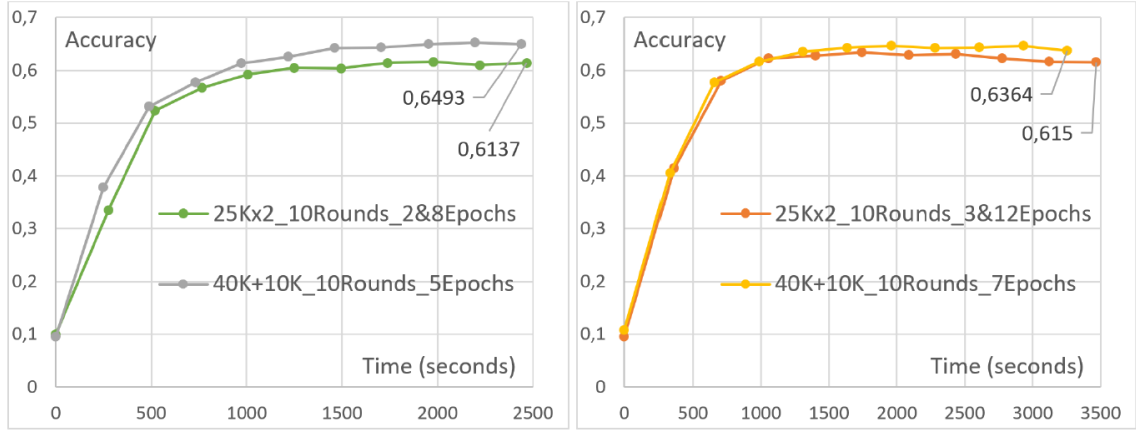


FIGURE 4.8: Accuracy for asymmetric data distribution vs. different numbers of local epochs, in time domain.

In our study, we intentionally introduced inhomogeneous and asymmetric data distribution to mimic real-world scenarios where data across users can vary significantly. Specifically in Figure 4.8, the yellow line corresponds to a balanced data distribution, where the dataset was evenly split between users (i.e., Raspberry Pis), with each user having 20,000 images. In contrast, the orange line illustrates an uneven data distribution. In this scenario, one user was allocated 10,000 images, while the other user had 40,000 images. This intentional variation allows us to evaluate the FL framework's ability to handle disparate data quantities among users. In all cases, the aforementioned trend is consistently observed. This underscores the efficiency of adapting asynchronous data splitting across heterogeneous FL clients, rather than modifying the overall training process to enforce synchronization of FL updates from diverse devices.

Another issue that frequently impacts the DNN performance is the concept of overfitting especially for the case of unbalanced datasets with a large number of local training epochs. While modifying the local training process for the case of FL with heterogeneous nodes, the data available at different nodes should be taken into account. When examining the final parts of the two simulations, in Figure 4.9, it is

possible to observe the effect of overfitting. The accuracy has somewhat lessened due to excessive training of the model on a small amount of data, leading to a decrease in universality for examples that were not utilized for training.

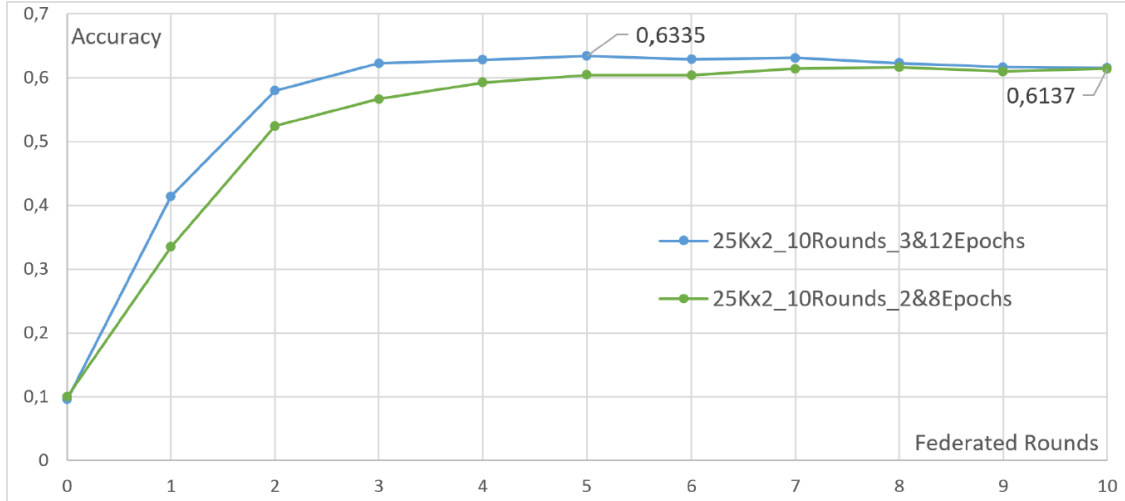


FIGURE 4.9: Overfitting for different numbers of local epochs.

It is worth mentioning that compensating for performance by distributing unequal amounts of data between devices is a challenging approach to implement. This is because it may not be possible to control the volumes of data collected by different clients. Conversely, increasing the workload of the highest-performing devices by increasing their local epochs can be easily achieved with communications from the server.

After defining the scenarios mentioned above, further exploration was conducted to determine the impact of a random selection of data compared to an ordered split without overlaps. It is important to note that the CIFAR10 dataset samples are inherently unordered, with images following one another irrespective of their class.

The initial approach involved dividing the dataset among multiple clients without repetition, resulting in a total of 50,000 samples between all clients, which is the complete training data. Imagine a circumstance where a pair of patrons are tasked

with handling the preparation assortment of CIFAR10, with one individual managing the initial 25,000 samples and the other in command of the residual 25,000 samples. These two data groups do not overlap, thus utilizing the entire training set.

Expanding this approach to five clients would require dividing the data set into five categories, each with 10,000 samples, with no overlaps or sample duplications. The second approach, however, was based on the random selection of samples from the dataset, allowing for a single data piece to be chosen several times within the same client. Moreover, data could be common to different clients, even if the sum of the samples from both clients exceeds the total number of samples from the training set. Thus, this method may yield a generally substandard model performance that aligns more with actual situations.

It is essential to consider that performance may vary from simulation to simulation, even with the same conditions, due to the random variables of these tests. As a result, the tests involving random samples were repeated multiple times, and the results presented represent their average. The impact of random selection, with identical data amounts and epoch numbers, is evident in Figure 4.10, which compares two centralized learnings.

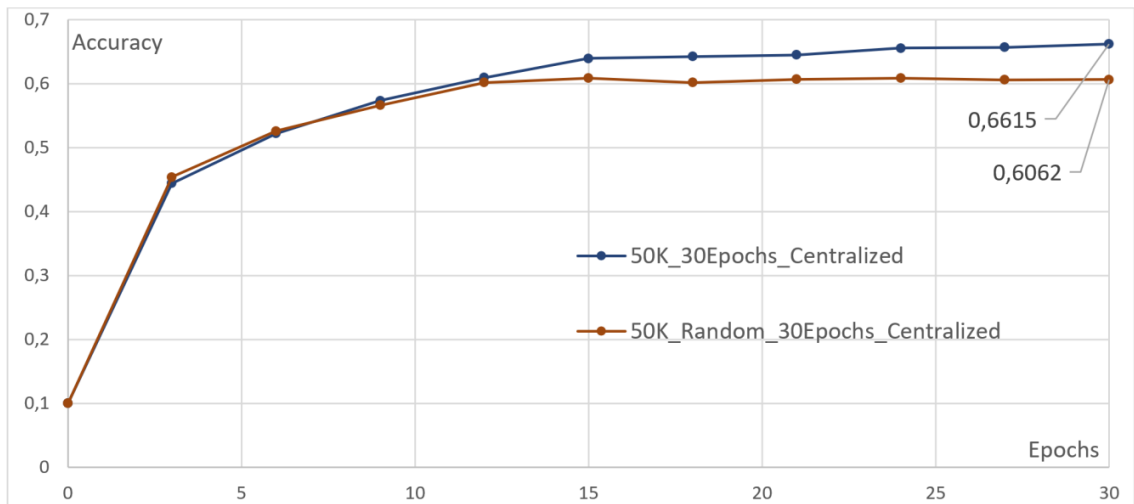


FIGURE 4.10: Centralized Learning with and without random data selection.

The aforementioned phenomenon is similarly observed in a federated scenario, as illustrated in Figure 4.11 by comparing two basic FLs involving two clients, each containing 25,000 images. The likelihood of encompassing a greater portion of the dataset and subsequently enhancing the ultimate accuracy increases as the number of randomly selected training samples increases.

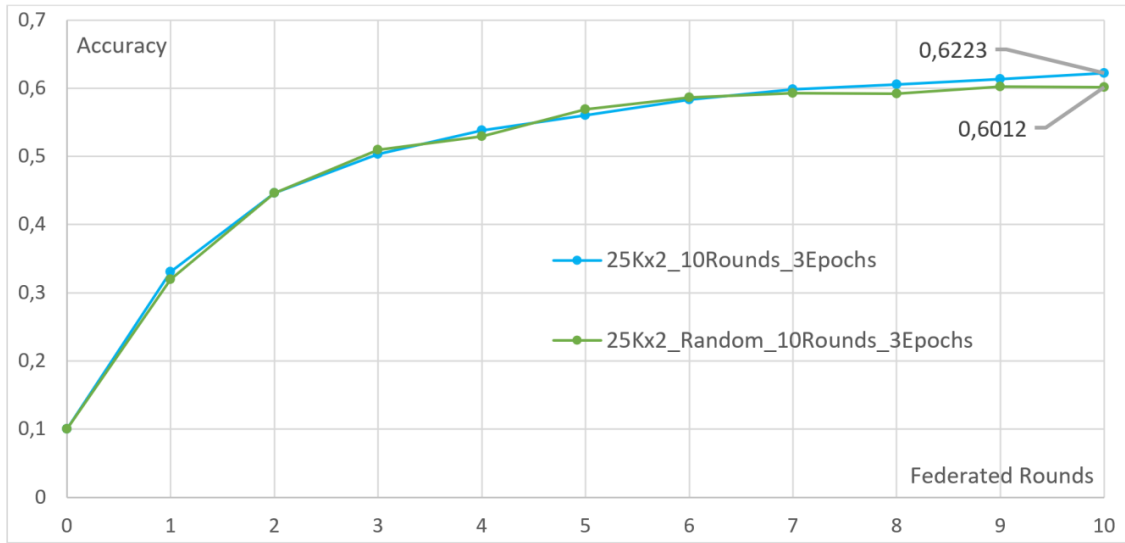


FIGURE 4.11: Federated Learning for 2 clients with and without random data selection.

As the quantity of randomly selected samples utilized for training increases, the probability of encompassing a more significant portion of the dataset also increases. This subsequently results in an enhancement of the final accuracy. This correlation is visually depicted in Figure 4.12, showcasing three FL evaluations conducted on two devices. The number of samples on each client has been progressively augmented from one simulation to the next, positively impacting the accuracy metric at the conclusion of each federated round.

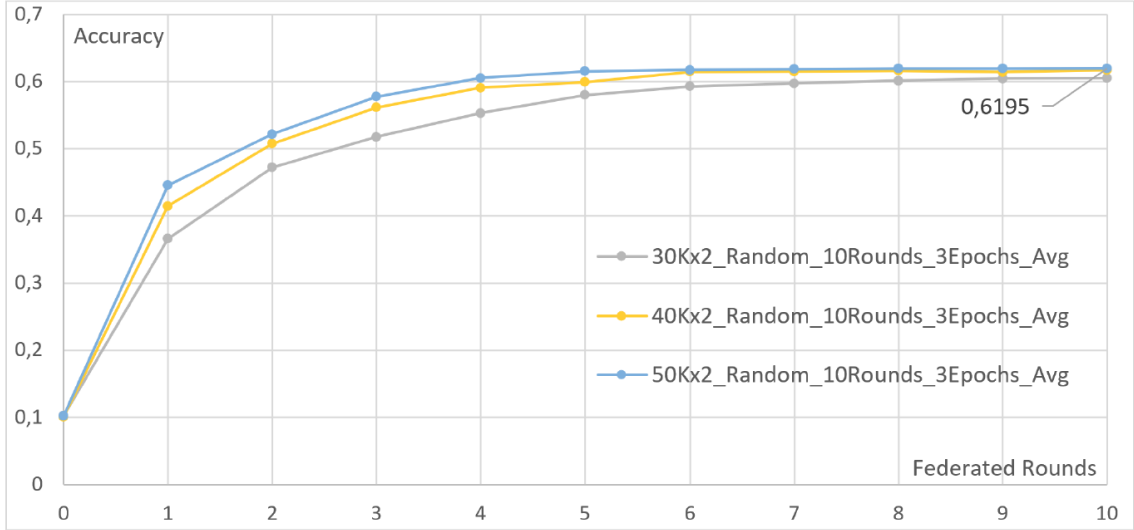


FIGURE 4.12: FL with different amounts of randomly chosen samples for two clients.

#### 4.2.6.3 Increasing the Number of Clients

The subsequent phase, aimed at enhancing the observance of FL and its scalability, entailed increasing the number of clients participating in the simulation to a maximum of 10. Of these, 2 were Raspberry clients, and 8 were virtual clients, coexisting with the server on the computer. The 10-client threshold could not be exceeded due to the restricted RAM on the laptop and the considerable resources required to train NNs. Nevertheless, this number of clients proved adequate in inducing a noteworthy decline in the model's performance as the number of devices involved escalated. In addition, it can be seen that the accuracy calculated by the server of the global model after each federated round is progressively sluggish as the number of clients used grows.

Figure 4.13 portrays the accuracy value for each federated round for five different simulations, each divided equally across all clients, using the internal training set (50,000 images). The specific parameters for each test are as follows: two clients

with 25,000 samples each, four clients with 12,500 samples each, five clients with 10,000 samples each, eight clients with 6250 samples each, and ten clients with 5000 samples each. The simulation employing only two clients is the best, whereas the test with ten devices yields the worst performance, requiring 40 federated rounds before achieving accuracy comparable to the former. However, it should be noted that, in the first case, Raspberries must operate with 25,000 samples each, resulting in relatively slower local periods, while in the second case, each client uses only 5000 samples, thereby completing its local epochs much faster.

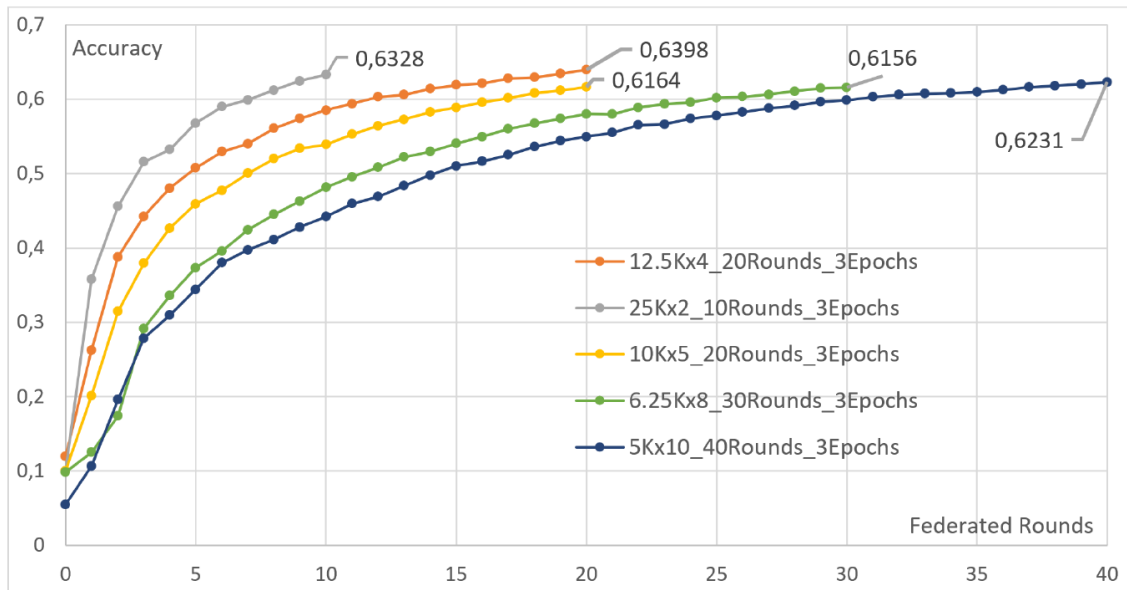


FIGURE 4.13: FL with different amounts of randomly distributed samples among different number of clients.

Examining Figure 4.14, representing the same performance while in the time domain, it is evident that the actual difference in performance between the simulations is much lower and sometimes even nonexistent. For instance, the test with ten clients completed its 40 rounds at a time, close to the tenth round of the test with only two clients. The same applies to the other curves, with the simulation involving four clients overlapping perfectly with that of two. This implies that, in the context of

a system with limited computational capacity, as represented by the Raspberry Pi, a federated approach generates relatively similar performance to centralized ones.

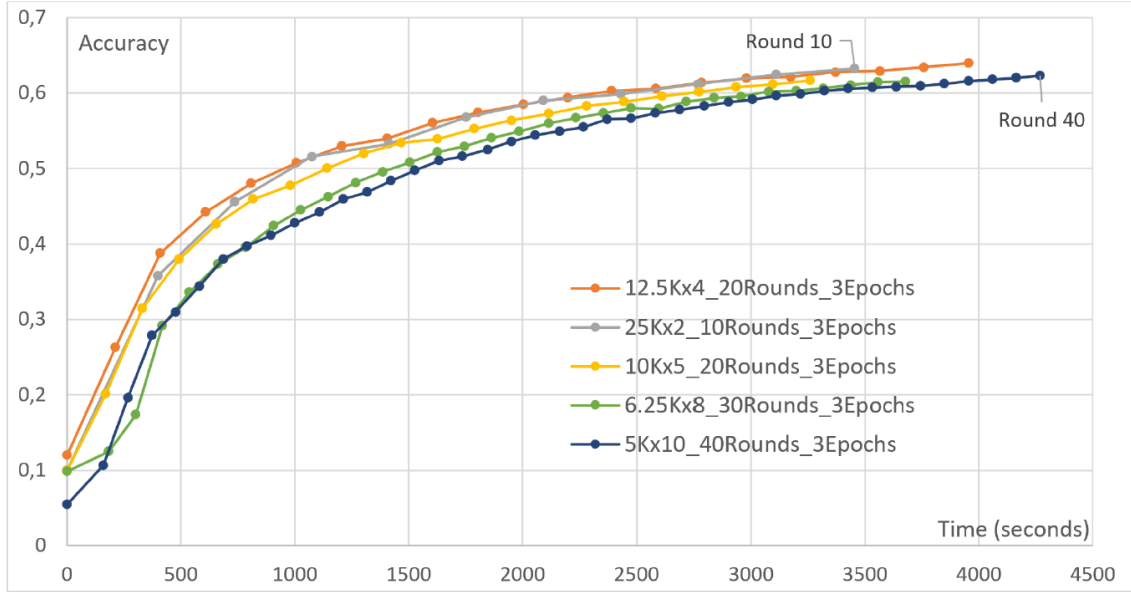


FIGURE 4.14: Accuracy vs. the number of rounds for different numbers of clients.

A similar pattern can be witnessed even with random samples, as illustrated in the two simulations depicted in Figure 4.15. The clients work with 30,000 and 12,000 images randomly selected from the dataset, and the algorithm trained with five clients obtains lower accuracy per round. However, since these simulations are completed twice as fast as the two-client simulation, the result is that the accuracy values are very similar moment by moment.



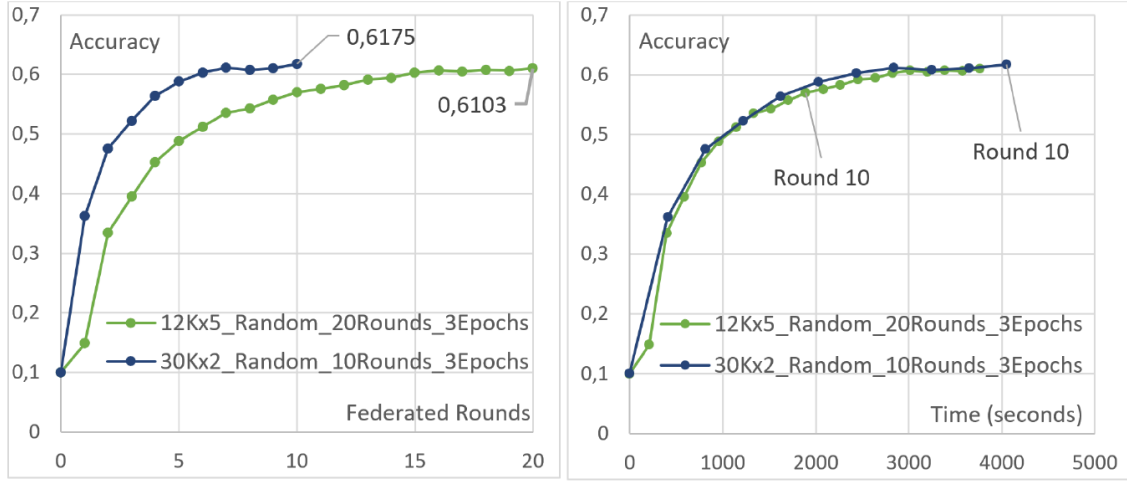


FIGURE 4.15: Accuracy with random samples and different numbers of clients.

#### 4.2.6.4 Effect of Pretraining

Upon completion of the previous phase, the training framework was further explored, emphasizing the introduction of pre-trained models. In all tests conducted thus far, the initial accuracy starts at approximately 0.1 (i.e., 10%), which aligns with a randomly initialized model. In this FL implementation, at the beginning of each simulation, the server randomly selects a client to transmit its original model, an NN with randomly assigned node weights, generating a very low accuracy of approximately 10%. This model is then disseminated from the server to all clients to establish a common starting point for training. Alternatively, it may be feasible to directly preserve a fundamental model on the server that is conveyed to all users during the initial stage. This method allows for using a pre-trained NN already on the server, resulting in enhanced performance for FL.

For this specific scenario, the model pre-training was conducted centrally on 30,000 samples randomly selected by CIFAR10, with data processing for 3, 6, or 9 periods, leading to three pre-trained models with varying levels of accuracy. This

approach observed how pretraining at different intensities contributes to FL's performance. As demonstrated in Figure 4.16, each test has a distinct initial accuracy value, followed by training through a straightforward implementation of FL with two clients, each with half a dataset and three local epochs per round. The impact of pretraining is discernible in the initial rounds, where trained simulations exhibit a significant advantage over cases without pretraining. This discrepancy weakens as the federated rounds continue until all curves converge around the tenth round, resulting in a final accuracy value that is relatively high, especially considering that the previous centralized benchmark on the internal training set had a score of 65%.

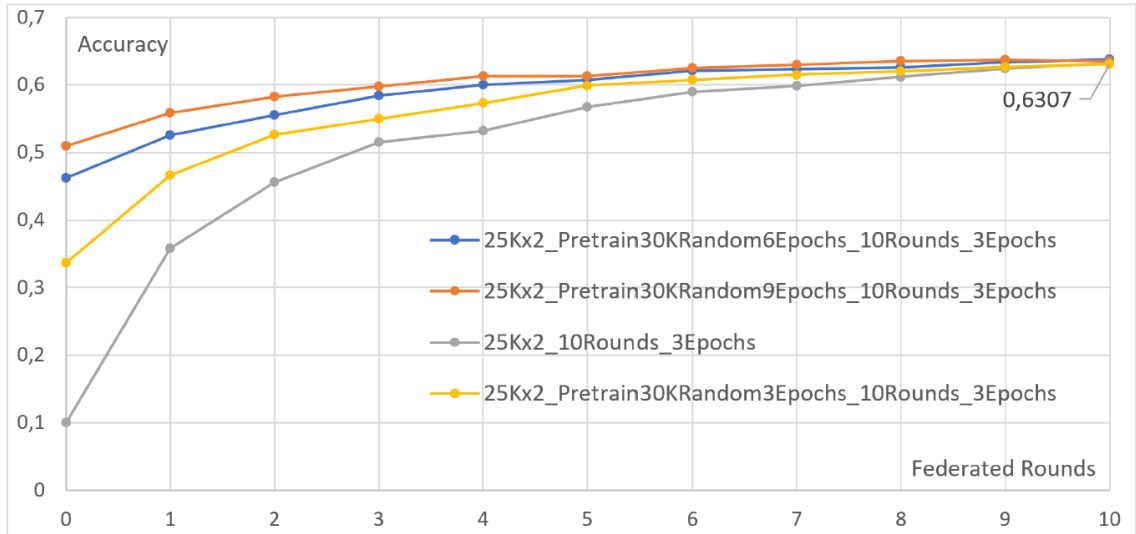


FIGURE 4.16: Accuracy with 2 clients and different pretraining levels.

Similar examinations were replicated in a federated context comprising five clients, each trained on a fifth of the dataset to extend the findings. Once again, as can be seen in Figure 4.17, a trend similar to the previous one is discernible, with a substantial effect of pretraining in the initial stages that diminishes until the curves converge. The incorporation of pre-trained models thus represents a commendable strategy to accelerate the FL of the network. This technique is particularly advantageous in situations with limited time available for training, as in this case, FL could

be stopped before converging to maximum accuracy, and pretraining would bring significant benefits, as observed in the first part of the graph.

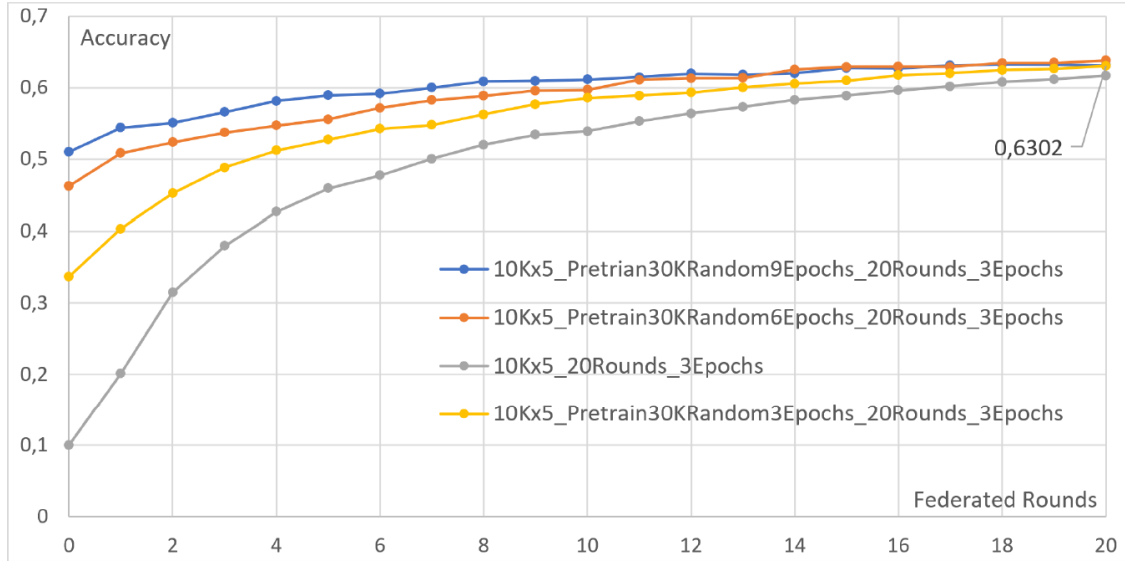


FIGURE 4.17: Simulations with 5 clients and different pretraining levels.

To further emphasize the impact of pretraining, FL was conducted again with 10 clients who had previously exhibited reduced performance compared to other simulations. However, this time a starting model already trained with 30,000 random samples for six periods was used. The outcome is shown in Figure 4.18, where the new simulation has substantially higher accuracy than the previous test with 10 clients and even surpasses the scenario with only two devices. This exemplifies how one can compensate for the decline in the accuracy of FL due to the increase in the number of devices involved by incorporating pretraining.

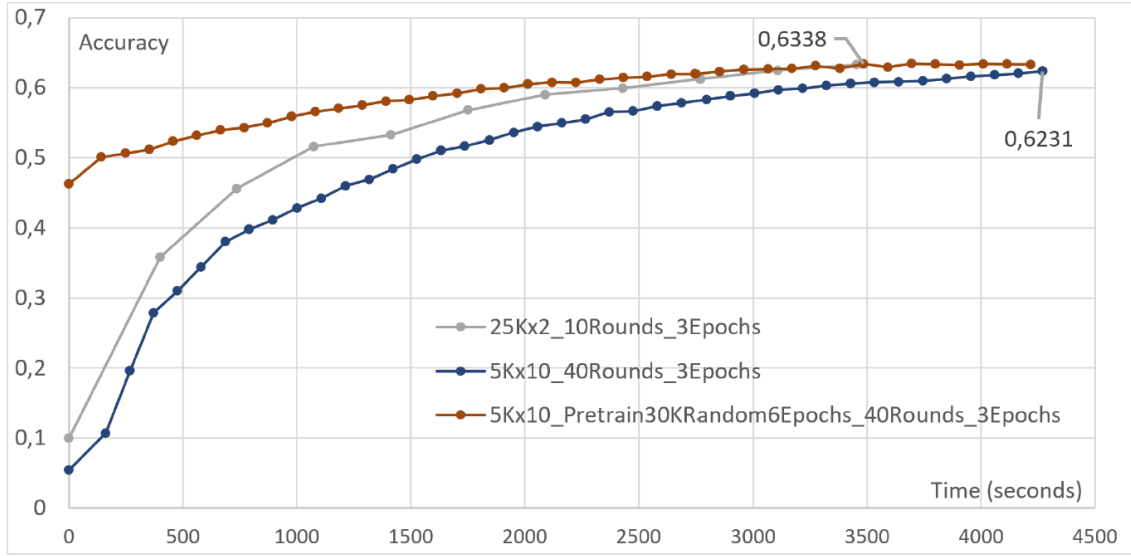


FIGURE 4.18: Two simulations with 10 clients with or without pretraining vs. simulation with 2 clients without pretraining.

However, we must recognize the deficiencies of this approach. Pretraining a model demands time and energy. Ideally, the optimal solution would be to conduct the pretraining directly on the server in a centralized manner. In this case, the reduced number of periods required to obtain a good starting model and the high computing capacity of the server would render the pretraining time almost negligible compared to the duration of the subsequent federated training. Nonetheless, this solution may not be feasible in a practical scenario as it implies the presence of a certain amount of training data directly on the server when, in several cases, FL is chosen precisely to circumvent the transfer of the local data collected by the various devices. A possible alternative could be the utilization of a more generic dataset for pretraining, then leaving clients with the task of integrating the model's specific features with their local data, akin to what occurs for TL.

#### 4.2.6.5 Transfer Learning

Drawing inspiration from the aforementioned analogy of TL, executing its functionality in the present framework has been feasible. However, it is imperative to specify that the measures implemented to achieve this outcome do not facilitate the attainment of the same level of automation as federated simulations. The practical principle is akin to pretraining, wherein an initial model is trained through centralized ML. Subsequently, this model is disseminated to various devices that conduct local training with their unique dataset.

In the first part of Figure 4.19, the fundamental model training is depicted, followed by the curves of multiple models obtained by the clients. The two models produced in the first scenario are almost identical, as both clients have 25,000 images of data and have been trained for the same number of epochs.

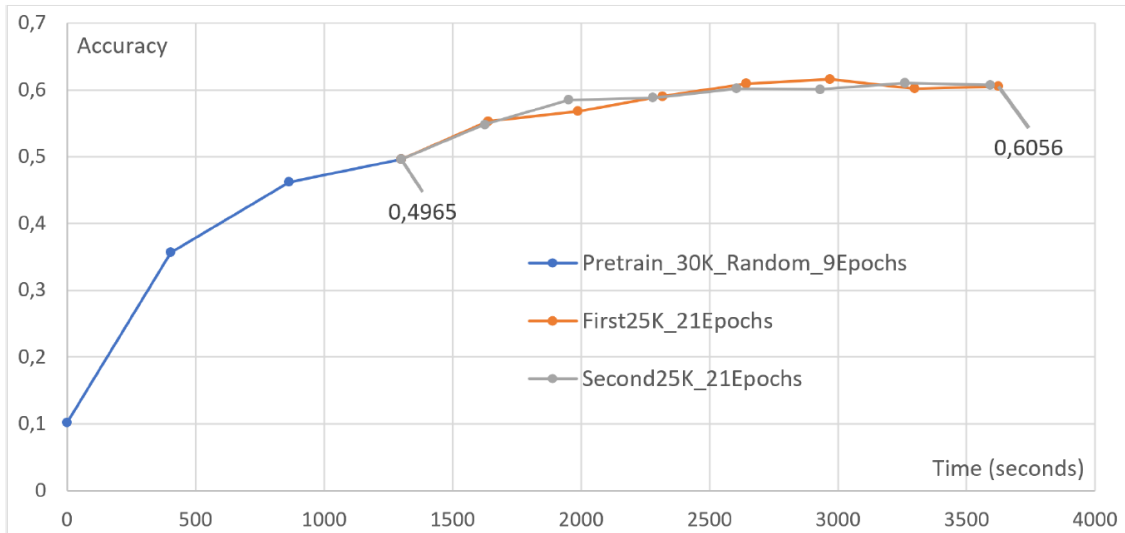


FIGURE 4.19: TL with 2 clients.

In contrast, in Figure 4.20, differences in terms of accuracy and training time between the three devices are apparent, which is expected since all the clients have trained for 21 periods. However, each client had varying sample sizes available, with the

first client having 10,000, the second having 20,000, and the third having 30,000 elements of CIFAR10. Thus, clients with more data will exhibit slower periods that guarantee greater accuracy.

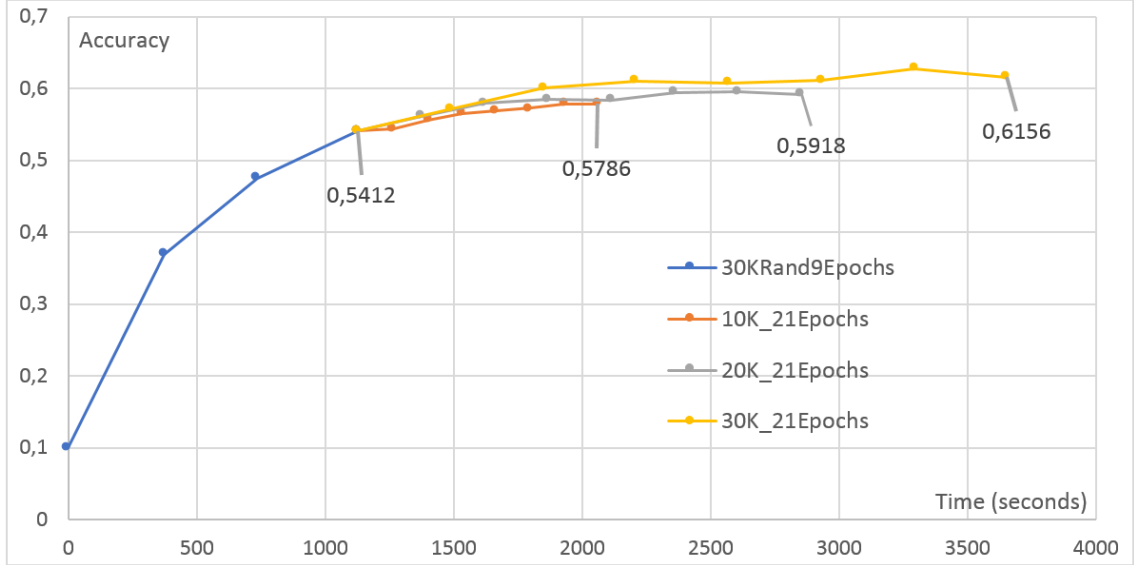


FIGURE 4.20: TL with 3 clients.

#### 4.2.7 Discussion

In this study, we conducted an in-depth exploration of FL methodology across heterogeneous nodes, offering critical insights into its application within the context of 6G technology. Our investigation revolved around a hardware-software integrated FL model, ingeniously leveraging a combination of Raspberry Pi devices and VMs as FL clients, each equipped with unique datasets sourced from the CIFAR10 dataset, a widely accepted benchmark in image classification. The experiments were meticulously designed to mirror real-world scenarios, addressing multifaceted challenges including varying computation resources, uneven dataset distributions, and the heating issues inherent in wireless devices.

A pivotal aspect of our research involved examining the impact of cooling mechanisms on training accuracy, elucidated in detail in Section 4.2.6.1. The insights garnered from Figure 4.5 underscore the significance of cooling devices in expediting model convergence, highlighting their practical relevance, especially in resource-constrained environments. Additionally, we delved into the complexities of heterogeneous client compensation, meticulously examining asymmetric data distribution scenarios, both with and without random selection, compared to a Centralized Benchmark, as delineated in Section 4.2.6.2 (Figures 4.6–4.8). These analyses illuminated the nuanced dynamics of FL performance, emphasizing the intricate balance required in distributing training data among disparate nodes. Furthermore, our study probed the issue of overfitting in FL, a critical concern often encountered in decentralized learning paradigms. Through Figure 4.9, we identified the challenge and subsequently addressed it by incorporating random selection strategies, showcased in Figures 4.10–4.12, thereby mitigating overfitting risks while optimizing model generalization. A comprehensive exploration into the scalability of FL was presented in Section 4.2.6.3, analyzing the impact of increasing the number of users on the system’s performance (Figures 4.13–4.15). This analysis provided valuable insights into FL’s scalability potential, crucial for its adoption in large-scale, dynamic environments. The effectiveness of pretraining in enhancing accuracy rates was explored in Section 4.2.6.4, revealing significant improvements showcased in Figures 4.16–4.18. Pretraining emerged as a powerful technique, elevating the model’s performance and showcasing its potential for optimizing FL outcomes. Finally, in Section 4.2.6.5, we delved into TL’s potential, evaluating its impact with varying numbers of clients (Figures 4.19 and 4.20). The results underscored TL’s capacity to enhance FL model performance, particularly when faced with diverse client configurations.

### 4.2.8 Conclusion

In this work, we have investigated the performance of the FL method including a group of heterogeneous nodes. In particular, a hardware-software integrated FL model is developed by using a set of Raspberry PI devices and VMs acting as an FL client with their datasets. Performance is analyzed for the case of an image classification problem with a widely known CIFAR10 dataset. Given the importance of distributed intelligence with heterogeneous wireless nodes in the upcoming 6G technology, a set of experiments are performed to analyze the FL performance in different cases. Main issues such as differing computation resources, uneven distributions of datasets, and heating issues of wireless devices were considered while performing the experiments. In addition, novel technologies such as the users of pre-trained networks for KT, were also considered. A more pro-analysis and concluding remarks are also presented during the discussion of simulation results. This study can be highly useful when considering the deployments of FL methods over heterogeneous 6G environments to enable large-scale, connected, cost-efficient, and reliable distributed intelligence.

In conclusion, our study's multifaceted approach, spanning cooling mechanisms, heterogeneous compensation strategies, overfitting mitigation, scalability analyses, and the integration of pretraining and TL, provides a holistic understanding of FL's dynamics across heterogeneous nodes. These nuanced findings not only contribute significantly to the academic discourse but also hold practical implications for real-world 6G deployments. By illuminating the complexities and offering viable solutions, our research empowers the seamless integration of FL in diverse, large-scale, connected, cost-efficient, and reliable distributed intelligence systems, laying the foundation for the future of intelligent wireless networks.



### 4.3 Real-World Implementation and Performance Analysis of Distributed Learning Frameworks for 6G IoT Applications

The imminent arrival of 6G technology heralds a new era marked by intelligence, full connectivity, and digitalization, achieved through the extensive deployment of distributed intelligent networks. This technological advancement is poised to provide a diverse array of intelligent services and applications tailored to specific demands. The success witnessed in 5G solutions underscores the pivotal role of IoT technology, which is expected to play an equally crucial role in the 6G society. As the IoT seamlessly integrates into various wireless scenarios, the 6G landscape anticipates the deployment of numerous sensory nodes capable of sensing their surroundings and generating copious amounts of high-quality data, crucial to enabling intelligent solutions in 6G networks.

ML plays a central role in the 6G vision, specifically in the empowerment of intelligent services. The deployment of various ML methods across different wireless scenarios is expected to facilitate intelligent solutions by analyzing 6G network data and extracting valuable insights. However, the conventional CL method, which concentrates training data on a single server, is inefficient when viewed from a 6G perspective. Given the need for next-generation networks to manage vast amounts of information that is predominantly generated at the network's edge, centralizing data incurs communication overheads and network traffic, limiting its applicability in the resource-constrained and time-critical 6G environment.

An efficient alternative to CL is DL, emphasizing decentralization of training. This

approach carries out the training phase directly on end devices or strategically positioned nodes at the edge of the network. DL enables the training of models by processing local data on each device, harnessing large amounts of heterogeneous data across the network. Beyond improving privacy and reducing communication overhead, DL distributes the computational load between devices, leading to improvements in energy management.

Within the realm of DL, FL has emerged as a successful approach to building high-quality ML models based on dispersed wireless data. The traditional FL approach involves devices with their datasets and a central server node. While offering advantages such as reduced data transmission costs and enhanced data privacy, FL faces challenges in the form of heterogeneous nodes with varying capabilities and the time required for model convergence, particularly in latency-critical 6G use cases.

Recognizing the challenges posed by limited client resources in the FL framework, TL is introduced as a strategic solution to fine-tune DNNs. Implementing FL directly on clients for all training processes becomes impractical due to resource constraints, making it challenging to achieve satisfactory accuracy. TL leverages pre-trained models and adapts them to the specific task at hand, offering an efficient hybrid approach that overcomes the limitations of resource constraints and time challenges in the 6G landscape. This incorporation of TL within FL promises to improve the feasibility and effectiveness of training DNNs in the context of 6G networks.

### 4.3.1 Technological Background

ML technology has attracted significant attention for empowering intelligent solutions in various wireless networks, spanning mobile communication networks [124], wireless sensor networks [125], transportation systems [47], and NTN networks [126]. It

proves instrumental in addressing intricate challenges within wireless communication, such as resource management [127], data offloading to edge networks [128], spectrum management [129], routing [130], and user server allocation [143], among others. Various ML techniques offer effective solutions to these complex problems. In particular, FL stands out as a widely adopted DL approach that provides efficient solutions. In a specific case, a paper [131] proposes energy-efficient FL solutions customized for wireless communication environments. Furthermore, research in [132] explores the applicability of FL solutions in smart city environments, demonstrating their versatility. The utility of FL extends to solving challenges within VNs, particularly in EC environments [144], digital twin-based VNs [145], resource allocation, and task offloading [146], and finds applications in diverse satellite networks [133]. Despite the extensive analyses of FL performance in various wireless settings, the existing body of work often lacks considerations for practical implementations.

Recently, various scholars have explored the implementation of a learning testbed that uses various sensory nodes to evaluate the performance of ML solutions. Raspberry Pi devices have become a common choice for assessing the efficacy of ML solutions that address various challenges in wireless networking. For instance, in [134], RL-based solutions were proposed to optimize routing processes in software-defined wireless sensor networks, with Raspberry Pi devices serving as sensor nodes for performance analysis. Another study [135] presented deep learning-based solutions for speed bump detection in intelligent vehicular systems, employing Raspberry Pi devices and associated camera modules for testing.

In the realm of healthcare monitoring, an approach called DIL was introduced in [136] to mitigate *catastrophic forgetting*. However, the application of this approach faces challenges on Raspberry devices due to the large model size (49 KB),

and the absence of details regarding data batch quality poses obstacles to convergence. Communication-efficient FL solutions, proposed in [137], were tested using Raspberry Pi devices in EC environments. Furthermore, the researchers in [44] delved into the analysis of FL performance in the presence of heterogeneous clients, providing valuable information for the integration of different IoT subsystems with heterogeneous nodes in the context of 6G networks.

FL has witnessed numerous advancements in recent years, extending its applicability to various domains and addressing various challenges. In particular, FL has been integrated with wireless channel properties, allowing devices to transmit model updates simultaneously and enabling automatic model aggregation through the wireless channel. This paradigm, known as unit-modulus over-the-air computation [147], improves the efficiency and scalability of FL in wireless communication settings.

Moreover, advancements in FL have led to the development of hierarchical FL frameworks tailored for complex applications, such as end-to-end autonomous driving [148]. These hierarchical approaches enable distributed model training across multiple layers of a hierarchical architecture, facilitating collaborative learning among interconnected components while ensuring efficient information exchange and decision-making in dynamic environments.

Additionally, recent research has explored the integration of task-oriented sensing, computation, and communication in FL settings [149]. By aligning sensing, computation, and communication resources with specific tasks or objectives, these integrated FL frameworks optimize resource allocation and coordination, thus improving the effectiveness and adaptability of FL systems in real-world applications.

Despite these contributions, to the best of our knowledge, the existing literature lacks a comprehensive study focusing on the implementation and performance analysis of a

real-world DL scenario with limited resources. Such scenarios require the utilization of TL for DNNs on end-users and clients, as exemplified by Raspberry Pis. This deficiency in the literature motivates our experimental analysis of the FL process in the presence of diverse sets of clients with varying resource capacities. This study aims to bridge the gap in understanding the practical implications and performance aspects of DL scenarios in resource-constrained environments, shedding light on the effectiveness of TL applications.

### 4.3.2 Contributions and Novelties

This study presents several significant contributions and novel insights into the realm of DL frameworks, particularly focusing on Federated Transfer Learning (FTL) and FL methodologies. The key contributions and novelties of our research are outlined as follows:

1. **Implementation and efficiency analysis:** We deploy FTL and FL frameworks on a real-world platform comprising heterogeneous devices, including Raspberry Pis, Odroid, and VMs. Through comprehensive efficiency assessments that include accuracy, convergence rate, and loss metrics, we demonstrate the superiority of FTL. In particular, FTL exhibits improved accuracy and reduced loss over shorter training durations compared to FL.
2. **Dynamic measurement and comparison of technical parameters:** Our study uses dynamic measurement techniques to evaluate key technical parameters, such as load average, memory usage, temperature, power, and energy consumption, in DL methodologies. The findings reveal that FTL exhibits

a lower average load on processing units and memory usage, while consuming less energy and power. This aspect is particularly crucial in emerging 6G scenarios characterized by resource-limited devices.

3. **Scalability analysis:** We perform scalability analyses to explore the impact of varying user counts and dataset sizes on the performance of the DL framework. Our results confirm the robustness and reduced sensitivity of the FTL to these variations, highlighting its adaptability and reliability in diverse operational scenarios.
4. **Innovative implementation strategies:** We introduce novel implementation strategies to address the practical challenges encountered during the implementation of the DL framework. In particular, we employ an asymmetric data distribution to closely emulate real-world scenarios, enhancing the applicability and robustness of our findings. Furthermore, we implemented cooling systems using fans and heat sinks to mitigate overheating and processing capability degradation, thus optimizing training performance and ensuring the reliability of our DL frameworks. Additionally, the introduction of memory swap functionality, which addresses the limited memory capacities of Raspberry Pis, represents a novel approach to enhancing the scalability and efficiency of DL frameworks.

### 4.3.3 Limitations

Although our study offers valuable information on the efficiency and performance of the FTL and FL frameworks, several limitations warrant acknowledgment. One notable limitation is the relatively limited number of users involved in our implementations. This constraint may restrict the generalizability of our findings to larger-scale

DL environments. Additionally, while the introduction of memory swap functionality represents a novel approach, further research is warranted to explore its implications on system performance and scalability in more extensive DL scenarios. Future research efforts could address these limitations by incorporating larger user cohorts and investigating alternative memory management strategies to further enhance the robustness and applicability of DL frameworks.

This work explores DL frameworks, specifically FL and FTL, and implements them on IoT devices within the context of 6G technology. The work begins with an introduction that outlines the objectives and necessity of DL frameworks in the evolving landscape of 6G and the IoT. Section 4.3.4 delves into the fundamentals of FL and extends the discussion to FTL, highlighting their applications and challenges. Following this, Section 4.3.5 details the implementation of the system, focusing on server and client configurations and functionalities. Experimental results and performance evaluations are presented in Section 4.3.6, with Experiment 1 analyzing DL with five heterogeneous clients and Experiment 2 exploring DL with three heterogeneous clients in order to analyze the scalability of the system. Through experimentation, the work demonstrates the superiority of FTL over FL in terms of performance and efficiency. Finally, the conclusion summarizes the key findings and contributions of the study, emphasizing the added benefits and importance of FTL in heterogeneous IoT environments.

### **4.3.4 Distributed Learning**

The traditional ML method functioned within a centralized framework, where distributed wireless nodes, acting as potential data origins, sent their data samples to

a central, more powerful node with substantial storage and computational capabilities. With increasing interest in the 5G system and the upcoming 6G technology, the wireless environment is filled with small devices capable of sensing the environment and producing vast amounts of high-quality data. Managing such a large volume of data using a conventional centralized approach could lead to significant communication overhead. Conversely, collecting data from multiple distributed nodes to create a comprehensive dataset at the central server node could result in notably higher training expenses. Furthermore, the emergence of new intelligent services and applications sets strict demands in terms of latency, privacy, and reliability, creating obstacles for centralized ML model training in wireless settings.

Recent progress in hardware and software technologies has significantly enhanced the onboard functionalities of end devices. With this enhanced capacity, these devices are now able to independently train complex ML models using their own datasets, removing the necessity of transmitting data over long distances and decreasing the extra training workload. Moreover, devices can interact with each other and server nodes to improve ML models, thereby boosting performance. This has given rise to a new approach in ML model training called DL, which seeks to overcome the constraints of conventional centralized techniques [20]. Different types of DL methods have been investigated recently.

There are two main strategies for implementing DL: distributing data in parallel or distributing models [138]. The former method involves spreading out the training data across multiple servers, while the latter method involves splitting the model's parameters among different servers. However, the complexity of dividing ML models into separate parameter groups poses a challenge when applying the parallel model approach. As a result, most distributed ML systems primarily rely on distributing data. Some prominent deep learning techniques include FL, collaborative learning,



MARL, and SL [17]. Among these, FL and its variations, like FTL, have become widely adopted in wireless networks due to their effectiveness in facilitating intelligent solutions.

#### 4.3.4.1 Federated Learning

FL acts as a decentralized structure for distributed ML, ensuring privacy by conducting learning directly on the end devices [150]. This deep learning approach allows the collaborative training of models across decentralized data sources by iteratively exchanging model updates while maintaining the raw data on individual devices. The FL algorithm, illustrated in Algorithm 4.2, functions in numerous rounds, with each round comprising several steps:

1. **Initialization:** At the beginning of the process, a global model ( $\theta$ ) is initialized. This global model serves as the starting point for training across all devices (Line 3).
2. **Iteration (round):** The algorithm iterates for a predetermined number of rounds. This iterative process aims to minimize a specific function for efficient FL [139]. Each round involves the following steps:
  - (a) **Device interaction loop:** Within each round, the algorithm interacts with each device (representing different data sources) individually (Lines 5–10).
    - **Broadcasting model:** The current global model ( $\theta$ ) is sent to each device participating in the process (Line 6).
    - **Local training:** Each device trains a local model ( $\theta_i$ ) using its own local dataset ( $D_i$ ) (Line 7). This ensures that models are trained using data that remain on the respective devices, preserving privacy.

- **Local model update:** After training, each device computes a local gradient update ( $\Delta\theta_i$ ) based on its local loss function ( $L(\theta_i, D_i)$ ) (Line 8).
  - **Sending update:** The devices send their respective gradient updates ( $\Delta\theta_i$ ) to a central server for aggregation (Line 9).
- (b) **Aggregation:** The central server receives the gradient updates from all devices and aggregates them to update the global model ( $\theta$ ) (Line 11). This aggregation step ensures that the insights gained from local data across all devices contribute to improving the global model. The update is performed using a learning rate ( $\eta$ ) to control the step size in the gradient descent process.
3. **End of iteration:** The algorithm repeats this process for a fixed number of rounds or until convergence criteria are met (Lines 4–12). Each round allows the global model to learn from diverse data sources without compromising individual data privacy.

This approach promotes privacy preservation, communication efficiency, and collaborative learning in ML tasks [151].

In FL, various algorithms were proposed to address different challenges and optimize the performance of the model on decentralized devices. Some commonly used FL algorithms include FedAvg, FedAvgM, FedIR, FedNova, FedCurv, MOON, and SCAFFOLD. Each algorithm has its unique characteristics and advantages, such as privacy preservation, communication efficiency, and robustness to heterogeneity. FedAvg, which is introduced in Algorithm 4.2 and we use in this study, is a widely adopted algorithm that averages local model updates to form a global model while mitigating privacy risks. FedAvgM extends FedAvg by incorporating momentum to

accelerate convergence [152]. FedIR focuses on addressing imbalanced data distribution among clients by adjusting the learning rate based on the inverse of the local gradient norm [153]. FedNova introduces adaptive learning rate scaling to enhance convergence in non-convex optimization problems [154]. FedCurv uses local curvature information to determine adaptive step size, improving optimization efficiency [155]. MOON and SCAFFOLD are more recent advances that aim to enhance communication efficiency and robustness to adversarial attacks, respectively [156]. It is important to note that the choice of the FL algorithm may significantly impact the performance and convergence behavior of FL systems, making algorithm selection a critical consideration in FL research and deployment. Further comparative studies are warranted on different FL algorithms to elucidate their relative strengths and limitations in various application scenarios.

---

**Algorithm 4.2** Federated Learning Algorithm

---

```

1: Input: Global model  $\theta$ , data subsets  $D_1, D_2, \dots, D_n$ 
2: Output: Updated global model  $\theta$ 
3: Initialization: Initialize  $\theta$ 
4: for  $t = 1$  to  $T$  do
5:   for  $i = 1$  to  $n$  do
6:     Broadcast  $\theta$  to device  $i$ 
7:     Train local model  $\theta_i$  on  $D_i$ 
8:     Compute local update  $\Delta\theta_i = \nabla L(\theta_i, D_i)$ 
9:     Send  $\Delta\theta_i$  to central server
10:  end for
11:  Aggregate updates:  $\theta \leftarrow \theta - \eta \sum_{i=1}^n \Delta\theta_i$ 
12: end for

```

---

To ensure the successful implementation of FL, it is crucial to thoroughly examine

several factors. These factors encompass the choice of learning devices, differences in performance levels between users, handling diverse training data, possible algorithms for combining local models, deciding on the aggregation approach on the server, and managing resource distribution [157].

When choosing devices, it is essential to assess factors like the quality and amount of local data, the efficiency of the connection to the central server, and the computational capacity of the client. In a diverse setting with devices of different onboard capabilities, it is crucial to consider the variations in computing power, as clients with limited capabilities could potentially hinder the overall learning process [158].

Two commonly accepted FL methodologies for DL are synchronous FL and asynchronous FL [159]. Synchronous FL involves all devices engaging in training local models concurrently, presenting difficulties in heterogeneous environments with diverse client capabilities. In contrast, asynchronous FL allows devices to train models at their individual speeds and transmit parameters to the server for aggregation, making it better suited for varying device capabilities.

In Section 4.3.6, an investigation is conducted into the effectiveness of various DL methods, with a specific focus on the application of asynchronous FL in compensating heterogeneous clients. This aims to overcome issues stemming from the varying performance capabilities of Raspberry Pis and VMs, with the goal of achieving equilibrium.

The variability in data between FL devices can have a notable impact on the effectiveness of FL [160]. In [141], the concept of federated continual learning was introduced to enhance performance with non-IID data, but the issue of scalability was not addressed. Convergence challenges may arise during the fusion of models

created locally, influencing the selection of an appropriate aggregation method. Conventional methods such as federated average (FedAvg) may demonstrate restricted efficacy, prompting the exploration of weighted means or device selection strategies guided by model performance.

These scenarios emphasize the necessity of establishing a suitable FL framework in diverse environments that are tailored to the features and conditions of the FL device. Employing a uniform FL approach may lead to diminished performance, underscoring the significance of evaluating the FL model's performance across different situations and opting for a suitable FL model [140].

#### 4.3.4.2 Federated Transfer Learning

Recent developments in the field of DL have elevated TL to a central position as an essential method for enhancing FL capabilities. The adoption of TL in FL is driven by the practical obstacles that arise when users confront resource constraints, making it impractical to perform extensive FL training on these devices. Integrating TL seeks to improve DNNs by leveraging existing knowledge, providing a tactical resolution to resource limitations experienced by users [57].

FTL is an extension of FL that incorporates the advantages of TL within a decentralized learning structure. In FTL, every client leverages their own data to develop a model, benefiting from the knowledge of a pre-existing global model instead of commencing the learning procedure anew. This method effectively reduces the need for computationally intensive training on separate devices, offering a valuable solution, especially in resource-constrained environments [161].

In the FTL algorithm depicted in Algorithm 4.3, the process begins with each client loading a pre-trained global model ( $\theta$ ) (Line 6). This pre-trained model contains

knowledge gained from previous tasks or domains. Subsequently, each client fine-tunes its local model ( $\theta_i$ ) on its respective local data subset ( $D_i$ ) using insights from the pre-trained global model (Line 7). The local model is then used to compute a local update ( $\Delta\theta_i$ ) based on the local loss function ( $L(\theta_i, D_i)$ ) (Line 8), which is sent to a central server for aggregation (Line 9). The central server aggregates these updates (Line 10) and updates the global model accordingly. This iterative process repeats for a predetermined number of rounds ( $T$ ), enabling collaborative learning between decentralized networks while leveraging the benefits of TL.

---

**Algorithm 4.3** Federated Transfer Learning Algorithm

---

```

1: Input: Pre-trained global model  $\theta$ , data subsets  $D_1, D_2, \dots, D_n$ 
2: Output: Updated global model  $\theta$ 
3: Initialization: Load pre-trained global model  $\theta$ 
4: for  $t = 1$  to  $T$  do
5:   for  $i = 1$  to  $n$  do
6:     Load pre-trained global model  $\theta$ 
7:     Fine-tune local model  $\theta_i$  on  $D_i$ 
8:     Compute local update  $\Delta\theta_i = \nabla L(\theta_i, D_i)$ 
9:     Send  $\Delta\theta_i$  to central server
10:   end for
11:   Aggregate updates:  $\theta \leftarrow \theta - \eta \sum_{i=1}^n \Delta\theta_i$ 
12: end for

```

---

In the context of FTL, the distinction lies in the initialization of the global model. Unlike FL, where the global model is typically initialized from scratch, in FTL, the global model is initialized with parameters pre-trained on a source domain dataset. This pre-trained global model serves as a starting point for learning in the federated setting. During the FTL process, the global model is collaboratively fine-tuned

across the decentralized devices using the available local data from the target domain. Using the knowledge encoded in the pre-trained global model, FTL aims to accelerate convergence and enhance performance, especially in scenarios where data in the target domain are limited or imbalanced.

In our work, Algorithms 4.2 and 4.3 illustrate the procedural flow of FL and FTL, respectively. Algorithm 4.2 outlines the steps involved in standard FL, where the global model is initialized without any prior knowledge of external datasets. Conversely, Algorithm 4.3 delineates the modified procedure for FTL, where the global model is pre-initialized with parameters pre-trained on a source domain dataset before being fine-tuned in the FL setting. In summary, while FL focuses on collaborative model training across decentralized data sources, FTL introduces the concept of TL by leveraging pre-trained models to enhance learning performance in federated settings. The distinction between “Global model” and “Pre-trained global model” underscores the initialization strategy employed in FL and FTL, respectively, reflecting their unique approaches to FL.

FTL presents several benefits [162]. To begin with, it overcomes the challenges related to limited resources by allowing clients to utilize a common knowledge base, removing the need for complex model training on individual devices. This speeds up the learning process and enhances the overall effectiveness of the FL framework [163]. Secondly, FTL enhances the model’s ability to generalize. By transferring knowledge from a pre-trained DNN to users, FTL establishes a foundational set of generalized features that can be adjusted based on each client’s specific data characteristics. This flexibility is crucial for building strong and efficient ML models [164]. Additionally, FTL aids in better model convergence, particularly in situations involving diverse devices [165]. The shared knowledge promotes a more coherent learning process, ensuring that models on different devices converge more effectively.

This is especially important in environments where there is significant variation in device capabilities [47].

In summary, the integration of TL into FL to create FTL presents a hopeful opportunity to address resource constraints and enhance the effectiveness, adaptability, and convergence of ML models in decentralized settings. The seamless merging of FL and TL concepts presents a holistic approach for DL in situations marked by varied and resource-limited users [166].

### 4.3.5 Implementation of the System

The main aim of this research is to develop, evaluate, and compare two detailed and systematically organized DL frameworks, known as FL and FTL, for training ML algorithms using a federated approach involving various distributed clients. Here, we outline the crucial steps taken in implementing both FL and FTL approaches.

The envisioned DL system utilizes a client-server setup, with a Windows PC acting as the server and a variety of Raspberry Pi and Odroid devices functioning as distributed users. Additionally, the integration of virtual clients enhances the distributed framework by introducing diversity. Incorporating both hardware and software clients into the federated framework offers significant benefits, especially in light of the importance of network programmability and virtualization in the realm of 5G/6G networks. Communication between devices is facilitated through the local network, with all hardware nodes connected via Wi-Fi.

The primary goal is to evaluate the effectiveness of these deep learning frameworks in settings with limited and varied resources. A standard task of image classification is conducted to develop a proficient DNN using the designated DL frameworks. The CIFAR10 dataset is used as a basis for training the model. It is important



to highlight the intentional creation of data diversity among FL clients by dividing the original dataset into different subsets. Although our study focuses on a specific ML task using predefined datasets, we recognize the potential of these frameworks to address general ML challenges. The implementation of FL and FTL frameworks is carried out using the Python programming language, with the PyTorch library for training the DNN model and the Flower library, designed for federated ML, to facilitate efficient automation of client–server interactions [142].

Later on, we elaborate on the complex setups utilized in both the client and server aspects of the examined distributed system. Figure 4.21 illustrates the essential elements of the DL framework under review, including a collection of Raspberry Pi and Odroid devices, VMs, and a centralized server responsible for consolidating parameters from the user community.

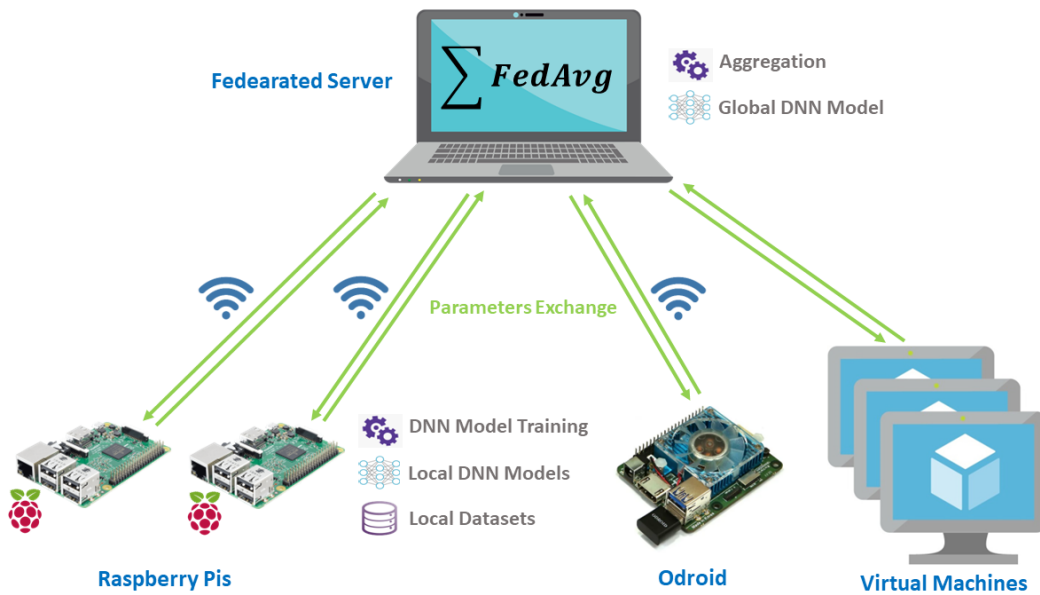


FIGURE 4.21: Considered distributed framework with heterogeneous clients.

#### 4.3.5.1 Server Configurations and Functionalities

In this section, we describe the fundamental procedures for setting up the server aspect of the FTL system and provide information on the software employed. The FTL server runs on a powerful Windows machine, namely an Asus K556U equipped with an Intel Core i7-6500U processor and 12 GB DDR4-2133 MHz RAM. Furthermore, it utilizes Intel Wireless-AC 9560 for Wi-Fi connectivity, which supports 802.11 a/b/g/n/ac standards.

The project opted for Anaconda as the development platform as it provides sophisticated package and library control for Python applications. The project utilized the standard Anaconda setup with Python 3.9.13 alongside PyTorch 2.0.0 and Flower 1.3, which were acquired and set up through the built-in terminal. The project architecture is based on three Python scripts: one for server operations, another for client tasks, and a third for specifying NN training procedures. Importantly, the system functions solely within the internal network, exchanging data among devices using the procedures outlined in the scripts.

The FL server plays various essential roles in facilitating FTL. Firstly, it initiates contact with a specified group of clients via the local network. Next, it sets up the global DNN model by incorporating pre-trained models tailored to particular situations. Afterward, it distributes the network parameters to all clients, ensuring a consistent foundation for federated training.

During the federated training procedure, participants train their individual models locally for a set number of iterations before sending their model parameters to the central server. The server combines these parameters to form a unified NN using the FedAvg algorithm. FedAvg is a privacy-conscious FL algorithm that trains a shared model among various participants while protecting the privacy of their data. The

process and mathematical representation of FedAvg can be found in Algorithms 4.2 and 4.3.

The performance of the combined model is evaluated based on accuracy and losses by utilizing a test dataset available on the server. The outcomes are saved in a CSV file. Subsequently, the improved combined model is sent back to the clients to start the next round of FL. This repetitive procedure persists until reaching the designated number of iterations, with each component playing a vital part in enabling the FL process.

#### **4.3.5.2 Client Configurations and Functionalities**

In this section, we provide more details on the variety of client devices used in our FTL framework, including Raspberry Pi devices, Odroid-N2L, and virtual clients created as VMs on a PC. Having a variety of client setups expands the range of capabilities and resources available in our IoT environment.

##### **Raspberry Pis**

Our client devices include Raspberry Pi 3B+ units, which were chosen for their cost-effectiveness and high customizability. The Raspberry Pi 3B+ features a Broadcom BCM2837B0 processor, a Cortex A53 (ARMv8) 64-bit SoC running at 1.4 GHz, 1GB LPDDR2 SDRAM, and 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless LAN. Additional specifications include Bluetooth 4.2, BLE, Gigabit Ethernet over USB 2.0, a 40-pin GPIO header, full-size HDMI, USB ports, DSI, a Micro SD slot, and a 5V/2.5A DC power supply.

To support our experiments, we installed the 64-bit Raspberry Pi OS, which is crucial for PyTorch compatibility. The configuration involved an SSH setup for the VNC

service, which allows remote desktop access. This facilitated real-time monitoring of simulations from a connected Windows computer, offering seamless interaction with the Raspberry Pi desktop.

The client-side code, executed in Python 3.9.2, utilized the PyTorch and Flower libraries. Clients ran the `client.py` script, inheriting contents from `cifar.py` for the generation, training, and evaluation of the NN. This configuration allowed efficient participation in the FTL process, optimizing training time and computational costs.

Here, we also take advantage of swap memory, also known as swap space, which is a section of a computer's hard disk or SSD that the OS uses to store inactive data from Random Access Memory (RAM). It involves moving data between the RAM and the dedicated space on the hard disk. When the RAM of a device like a Raspberry Pi is not enough to handle all the running processes, the OS can move some of the data to the swap space. This allows larger and multiple processes to run concurrently. The kernel may also decide to move data that has not been touched for a while to the swap space, freeing up RAM for other applications or cache. This makes better use of the available RAM and can improve system performance. However, using swap has its disadvantages; accessing data from the hard disk is significantly slower than accessing it from RAM. This can lead to slower response times for the system and applications if processes are too aggressively moved out of memory. We have made 3GB of swap memory for the Raspberry Pis.

## **Odroid**

We introduce Odroid-N2L devices, which feature a quad-core Cortex-A73 (2.2 GHz) and dual-core Cortex-A53 (2 GHz) Amlogic S922X processor with ARMv8-A architecture. Equipped with 4GB of LPDDR4 RAM, these devices have enhanced computational capabilities compared to Raspberry Pi devices. Their inclusion adds

diversity to our client pool, accommodating various resource capacities within the IoT ecosystem.

### **Virtual Clients**

In addition to physical devices, we include as many as two virtual clients created as VMs on a personal computer. Although all virtual clients were set up on a single PC for testing purposes, the system can be expanded to multiple-PC setups to enhance client variety. These virtual clients, utilizing the same scripts as the physical clients, provide additional computational capabilities [167]. Each of the virtual clients was configured to utilize one of the CPU cores of the hosting hardware, which was an ASUS K556U Core i7 2.7 GHz with 12 GB RAM memory. The virtual clients were allocated access to all system resources, ensuring consistent performance and enabling seamless integration with the FL framework. The specifications of the hardware used in this experiment are summarized in Table 4.1, which compares the capabilities of all different client types, including CPU speed, RAM size, storage capabilities, and other relevant specifications.

In order to avoid overheating in Raspberry Pi devices, a cooling fan was utilized throughout all experiments to maintain peak performance. The efficiency of this cooling method is further investigated in Section 4.3.6.

TABLE 4.1: Hardware specifications

| HW     | RAM                  | SWAP | CPU  | No.<br>of<br>Cores | Memory                |
|--------|----------------------|------|--|--------------------|-----------------------|
| PI     | 1 GB LPDDR2<br>SDRAM | 4 GB | Cortex-A53 (1.4 GHz)   | 4                  | 16 GB<br>Micro SD     |
| Odroid | 4 GB LPDDR4<br>RAM   | 0    | Quad-core Cortex-A73<br>(2.2 GHz), Dual-core<br>Cortex-A53 (2 GHz) | 6                  | 32 GB<br>eMMC         |
| PC     | 12 GB DDR4<br>SDRAM  | 0    | Core i7-6500U (3.1 GHz)  | 2                  | 1 TB<br>Hard<br>Drive |

In our FTL framework, the client nodes perform a series of functions. Communication channels are established with the server, global model parameters are received at the beginning of each round, and local training is performed on client devices. These trained models are evaluated locally, and the metrics are transmitted back to the server. The entire process is detailed in Figure 4.22, providing insight into the experimental setup that uses Odroid and Raspberry Pi nodes, VMs, and an FL server installed on a Windows PC.



FIGURE 4.22: Experimental setup used during the first DL implementations.

### 4.3.6 Experimental Results and Performance Evaluations

In this section, the performance analysis of the proposed FTL framework is performed using the Python simulation environment. The evaluation focuses on image classification, using the CIFAR10 dataset, which comprises 60,000 colorful pictures with a resolution of  $32 \times 32$  pixels, classified into 10 classes, each containing 6000

images. The dataset is partitioned into 50,000 training samples and 10,000 verification samples. A random arrangement of images is ensured while maintaining a uniform distribution of classes. We experiment in two different settings, one with five heterogeneous clients and the other with three heterogeneous clients.

#### 4.3.6.1 Experiment 1: DL with Five Heterogeneous Clients

The first experimental setup involves one Odroid, two Raspberry Pi devices, and two virtual clients, as shown in Figure 4.22. Each virtual client is assigned an equitable share of CPU resources on the host machine, ensuring a fair and consistent experimental environment. This resource allocation strategy guarantees a balanced and representative simulation, devoid of the influence of uneven resource distribution among virtual clients, providing a reliable basis for experimental findings. The initial distribution of the CIFAR10 training set among clients is followed by 30 federated iterations, comprising three local training epochs. The test data, consisting of 10,000 CIFAR10 samples, are used throughout the simulation.

The accuracy of the model is measured using a metric that calculates the percentage of correct predictions out of all predictions. The accuracy values presented are normalized between 0 and 1. To accelerate the training process, the number of training iterations and the overall size of the data are limited, setting an upper bound on DNN performance, specifically ResNet in our case. The experimental results indicate that with FTL, the DNN achieves an accuracy of up to 83%, compared to approximately 60% with FL. Improved performance can be achieved with additional resources such as data samples, devices, and training iterations.

Figure 4.23 illustrates the performance of the FTL and FL frameworks in basic settings, where users possess varying amounts of uncorrelated data and onboard



capabilities. The simulation comprises 10% of randomly selected data for Raspberry Pi 1, 15% for Raspberry Pi 2, 25% for Odroid, 20% for Virtual Machine 1, and 30% for Virtual Machine 2, totaling five clients. Both the FTL and FL models undergo training for 30 global federated rounds. The results reveal that the FTL framework surpasses FL in terms of performance and accuracy, achieving higher accuracy in a shorter time, indicative of its faster DNN training capabilities.

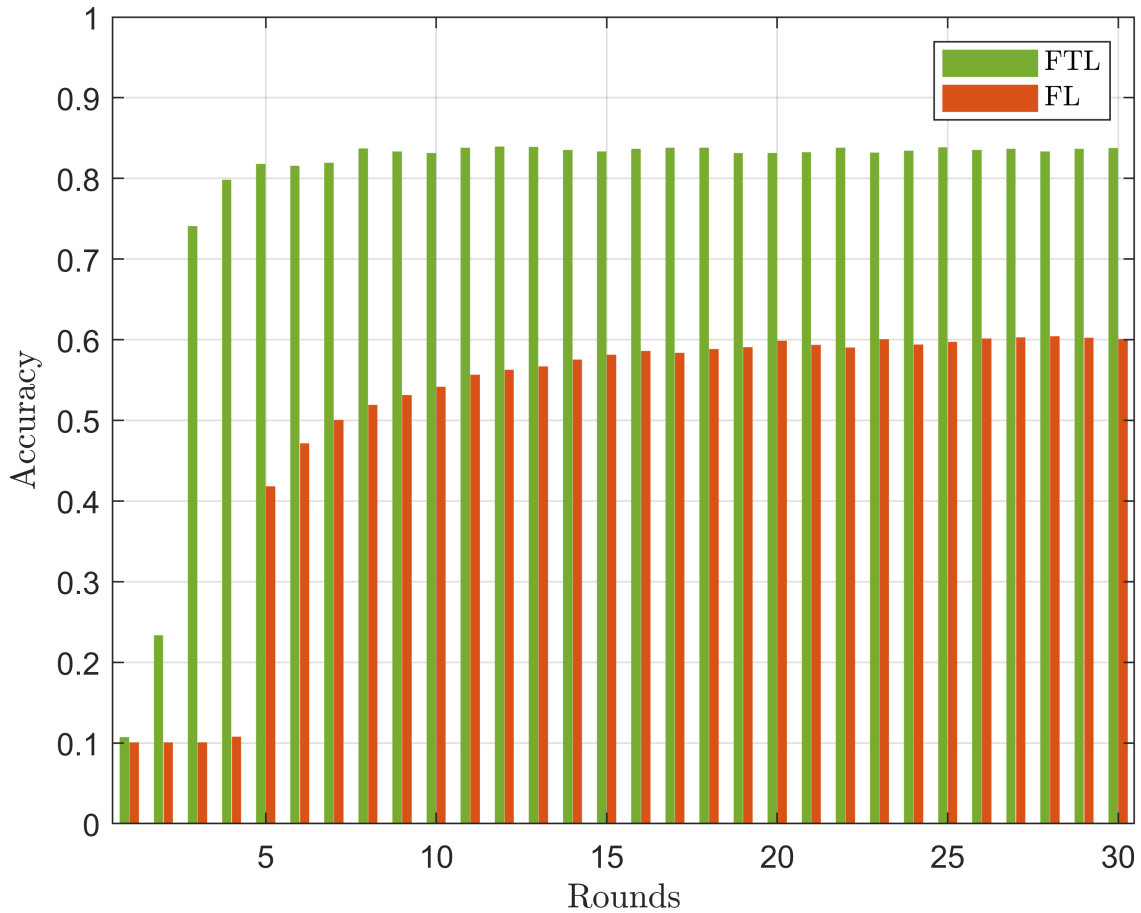


FIGURE 4.23: Accuracy of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. rounds.

To quantify this superior performance in terms of latency, a settling time of 2% is considered, representing the time it takes for accuracy to reach or fluctuate within 2% of the steady-state accuracy after 30 rounds. Figure 4.24 illustrates a settling time of approximately 9720 s for FTL compared to 27,300 s for FL, indicating a

three-fold faster convergence for FTL. Moreover, FTL exhibits potential advantages in terms of reduced communication overhead and enhanced data privacy, which are crucial aspects in emerging wireless scenarios, particularly in 6G.

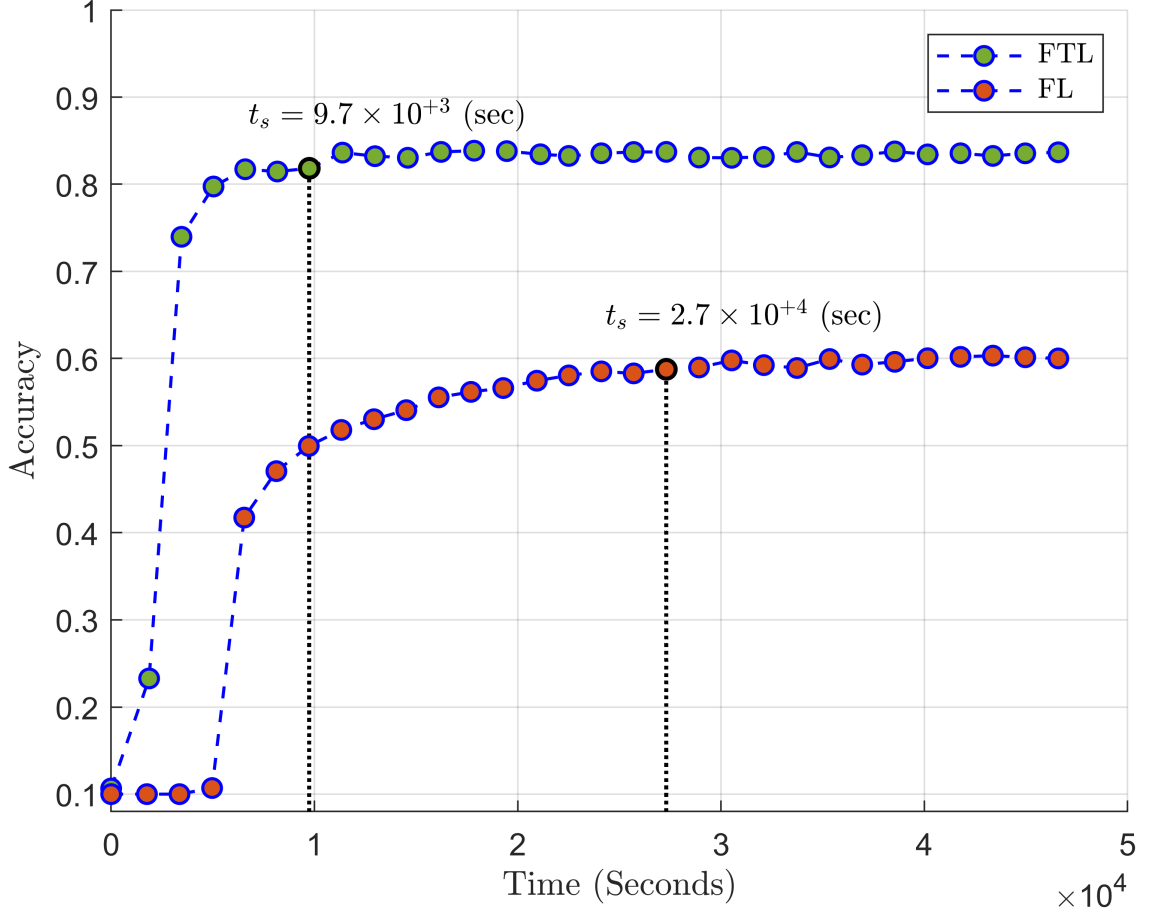


FIGURE 4.24: Accuracy of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. time.

In the case of FTL, the global model is pre-trained using a large dataset from a related domain, such as ImageNet, before being fine-tuned on the federated dataset. This pre-training step initializes the global model with knowledge learned from the source domain, allowing it to capture generic features and patterns that can facilitate faster convergence and improved performance when adapting to the target federated dataset. In our experiment, we employed a ResNet DNN architecture, which is commonly pre-trained on large-scale image datasets like ImageNet due to

its effectiveness in feature extraction and representation learning. By initializing the ResNet model with weights learned from ImageNet, the model already possesses a strong foundation of feature representations, enabling it to learn more efficiently and effectively from the federated CIFAR10 dataset during the fine-tuning phase. This approach leverages the transferability of features learned from ImageNet to the task of classifying CIFAR10 images, ultimately leading to enhanced performance and accelerated convergence in the FL process.

In our implementation, we used the categorical cross-entropy loss function as a metric to assess the performance of the FTL framework against conventional FL. This loss function measures the difference between the true probability distribution of the class labels and the predicted probability distribution generated by the NN model. Mathematically, the categorical cross-entropy loss  $L(y, \hat{y})$  for a single sample is defined as

$$L(y, \hat{y}) = - \sum_i y_i \log(\hat{y}_i)$$

Here,  $y_i$  denotes the  $i$ th element of the true class distribution vector  $y$ , and  $\hat{y}_i$  represents the  $i$ th element of the predicted probability distribution vector  $\hat{y}$ .

Next, in Figures 4.25 and 4.26, the loss incurred by FTL across five clients—two Raspberry Pis, one Odroid, and two VMs—is compared to that of the conventional FL approach over multiple training rounds and versus time. Our analysis reveals a markedly superior performance of FTL in terms of loss minimization. Specifically, an examination of these figures elucidates that the steady-state loss attained by FTL is merely one-third of that encountered with the FL methodology. Consequently, our approach significantly advances towards the objective of minimizing the loss cost function inherent in this multi-class classification problem facilitated by DNNs.

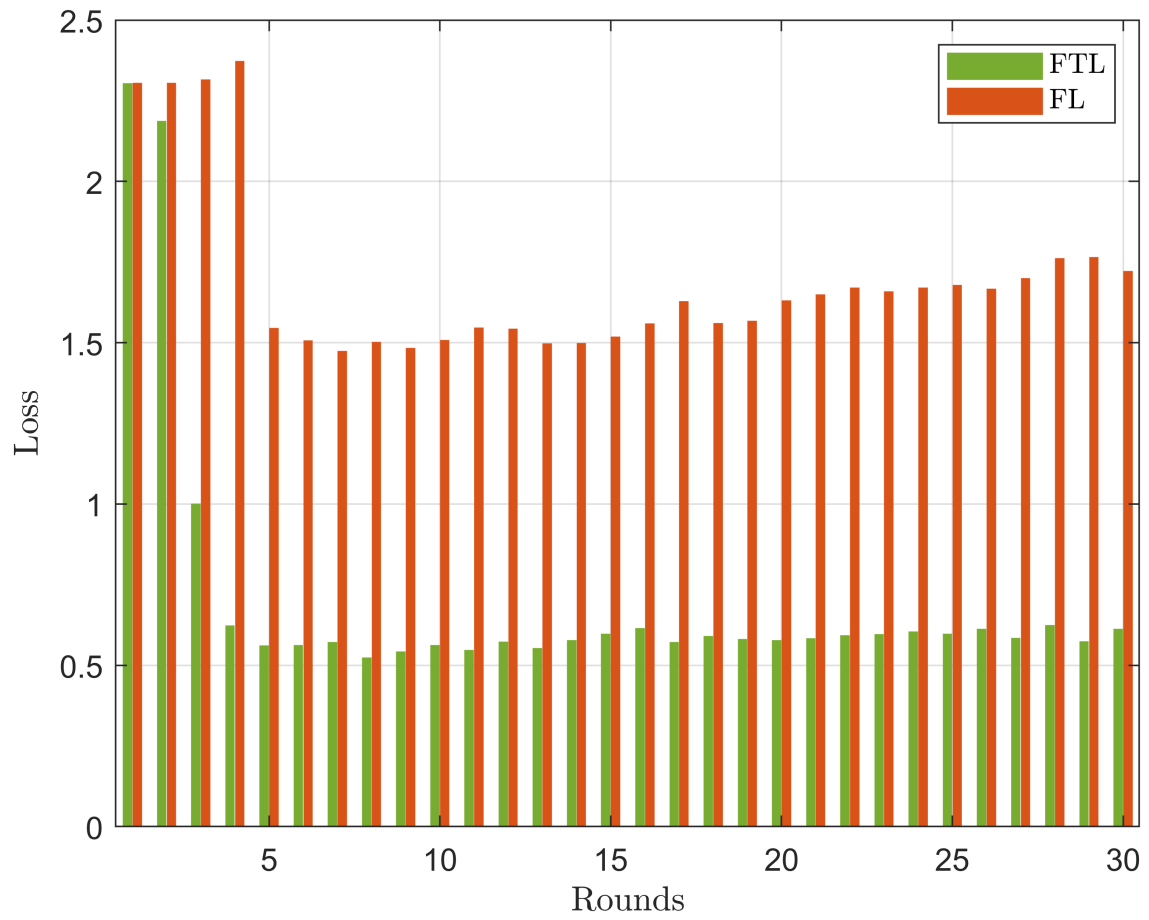


FIGURE 4.25: Loss of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. rounds.

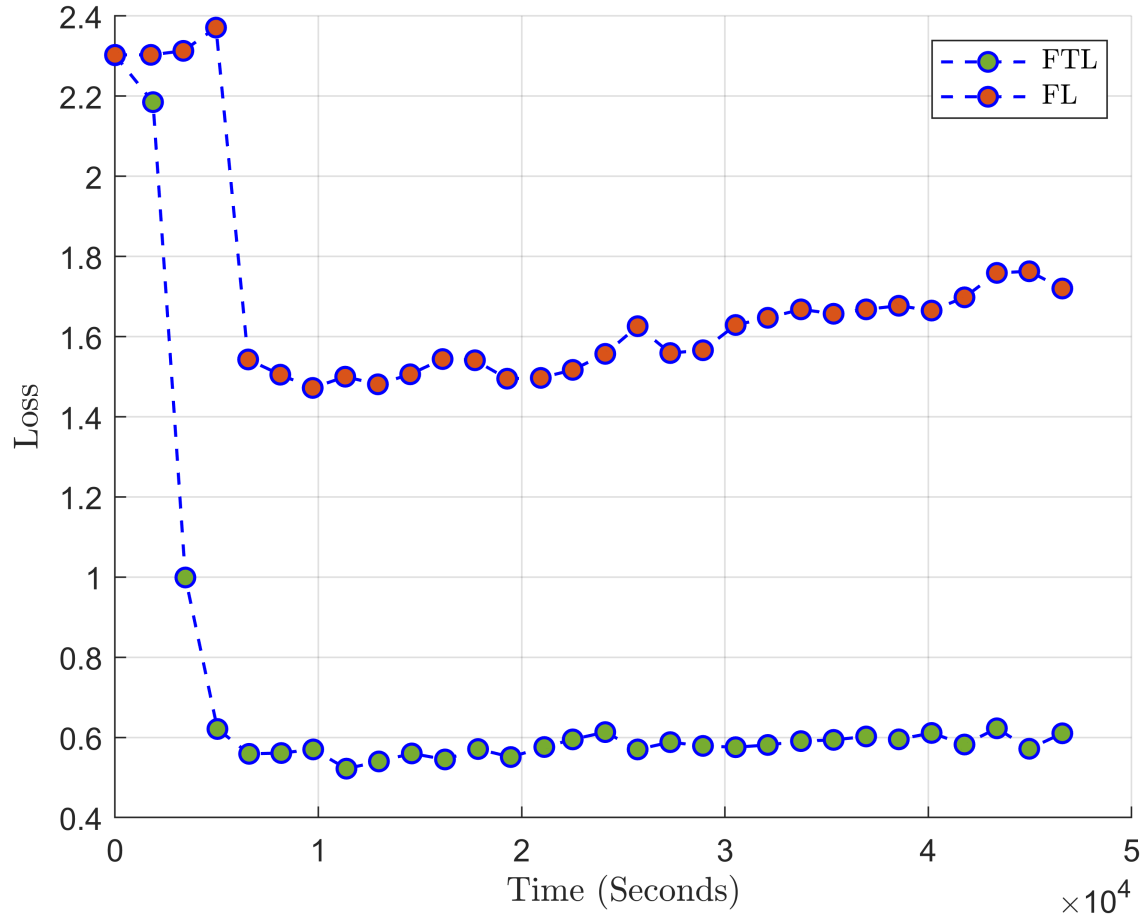


FIGURE 4.26: Loss of FTL for 5 clients, 2 Raspberry Pis, 1 Odroid, and 2 virtual machines, compared to ordinary FL vs. time.

Subsequently, we utilize *htop*, a command-line utility renowned for its real-time monitoring capabilities of system resources and server processes, to dive deeper into dynamic parameters such as memory usage, average CPU load, and temperature. Using *htop*'s interactive interface, we obtain crucial insights into the operational status of our distributed clients.

Our initial focus is to examine the average load exerted on the processing units of the client devices. The load average, a metric indicative of the computational workload borne by the CPU, is of paramount interest. For example, a load average of 1.0 on a single-core CPU signifies full utilization, while a load average of 2.0 on a

dual-core CPU equals 100% CPU usage. Comprising three values—a one-minute average, a five-minute average, and a fifteen-minute average—the load average affords a comprehensive snapshot of CPU utilization trends over varying time intervals. We meticulously measured these load average values across different clients, including Raspberry Pis and Odroid, culminating in the generation of error bar charts, as depicted in Figure 4.27.

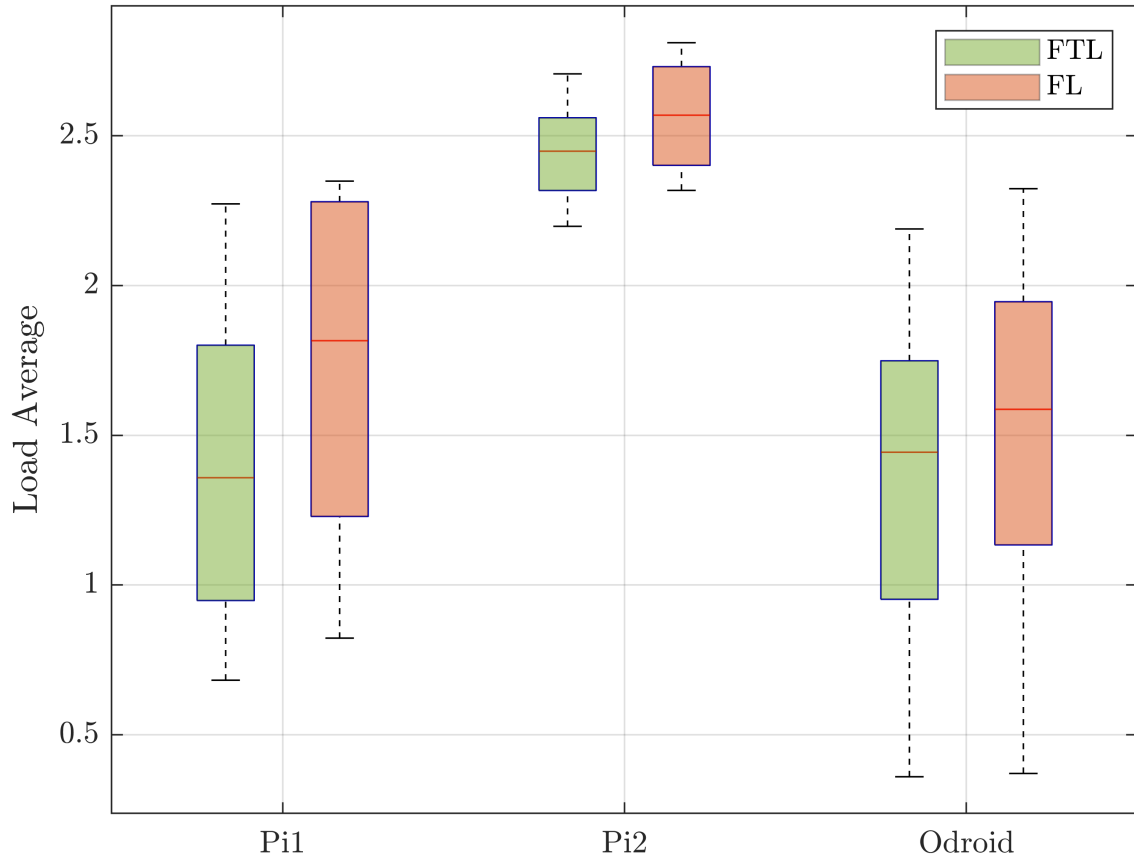


FIGURE 4.27: Average load on the Raspberry Pis and Odroid when running FTL and FL with 5 clients.

In our experimental setup, we utilized Raspberry Pis with four cores and Odroid with six cores. These hardware configurations significantly influence the interpretation of load average values, as depicted in Figure 4.27. Load average, as a metric indicative of CPU workload, becomes more comprehensible in light of the processor counts

of the client devices. For instance, a load average of 1.0 on a single-core CPU indicates full utilization, while the same load average on a quad-core Raspberry Pi suggests only a quarter of the CPU resources are utilized, and on a six-core Odroid suggests one-sixth of the CPU resource utilization. Therefore, understanding the processor count of each device is crucial for contextualizing the load average values accurately. By meticulously measuring load average values across different clients and considering their respective processor counts, our analysis aims to provide a nuanced understanding of CPU utilization trends and their implications for FL performance.

In particular, our observations unveiled lower load average values when executing FTL on clients as opposed to employing FL to train ResNet models. This disparity can be attributed to the utilization of pre-trained networks within the FTL framework, obviating the need for initialization and training from scratch, thereby alleviating the computational burden on the CPU.

A nuanced examination of the load average dynamics across different client devices further elucidated intriguing trends. In particular, the load average exhibited a descending order from Odroid to Raspberry Pi 1 and Raspberry Pi 2. This order can be attributed to the varying processing capabilities inherent in each device. Specifically, Odroid, characterized by superior computational capacity compared to Raspberry Pis, demonstrated the lowest load average due to its prompt completion of training tasks. On the contrary, Raspberry Pi 2, tasked with a larger dataset for ResNet training, endured a prolonged training duration compared to its counterparts, consequently registering a higher load average.

In Figure 4.27, the load average is defined based on the percentages of CPU core usage, which reflect the system's utilization over a period of time. Since the VMs in our study were implemented on the same CPU, each assigned to a distinct single

core, the load average parameter becomes less relevant for VMs as their resource utilization is inherently tied to the CPU core they are allocated to. Therefore, including VMs in the load average plot would not provide meaningful insights into their resource utilization patterns.

Adjacent to our investigation of the dynamic temperature profiles of devices during DL training sessions, as depicted in Figure 4.28, a discernible correlation emerges between the temperature dynamics and the distribution of workload and computational resources between devices. In particular, the observed trend closely parallels that identified in the measurement of load averages, indicating a coherent relationship between computational workload, training efficiency, and thermal management.

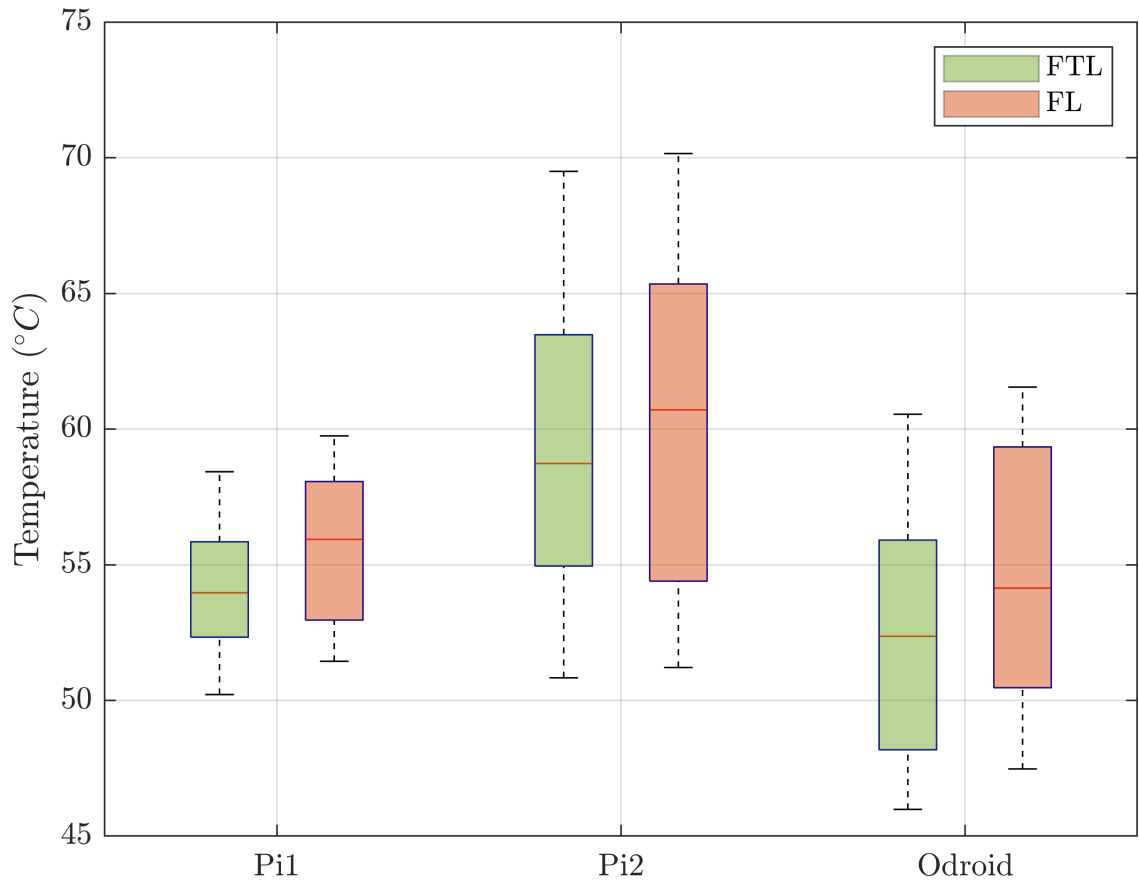


FIGURE 4.28: Raspberry Pis and Odroid temperature when running FTL compared to FL with 5 clients.



Specifically, devices allocated with lighter training burdens demonstrate faster training completion times, giving them ample opportunity to dissipate heat and maintain cooler operating temperatures. This can also be attributed to the fact that the Odroid device has a built-in cooling mechanism of a fan and a heatsink. Consequently, the incidence of overheating is significantly mitigated, underscoring the importance of workload optimization to promote thermal stability within DL environments.

Furthermore, our analysis reveals a notable distinction in the temperature dynamics between the FTL and FL methodologies in identical client configurations, data distributions, and environmental conditions. Interestingly, devices that execute FTL exhibit lower operating temperatures compared to those that implement FL. This disparity underscores the efficacy of FTL in minimizing thermal stress on client devices, thus enhancing operational reliability and longevity.

Subsequently, we conducted an assessment of memory utilization across various client devices, encompassing Raspberry Pis, Odroid, and VMs. The results of this investigation are depicted by error bar graphs, as illustrated in Figure 4.29. As expected, the disparity in memory usage values between the FL and FTL frameworks is not pronounced. However, FTL exhibits a discernible alleviation of memory burden, attributable to intervals of inactivity following each round and the utilization of pre-trained parameters rather than commencing training anew.

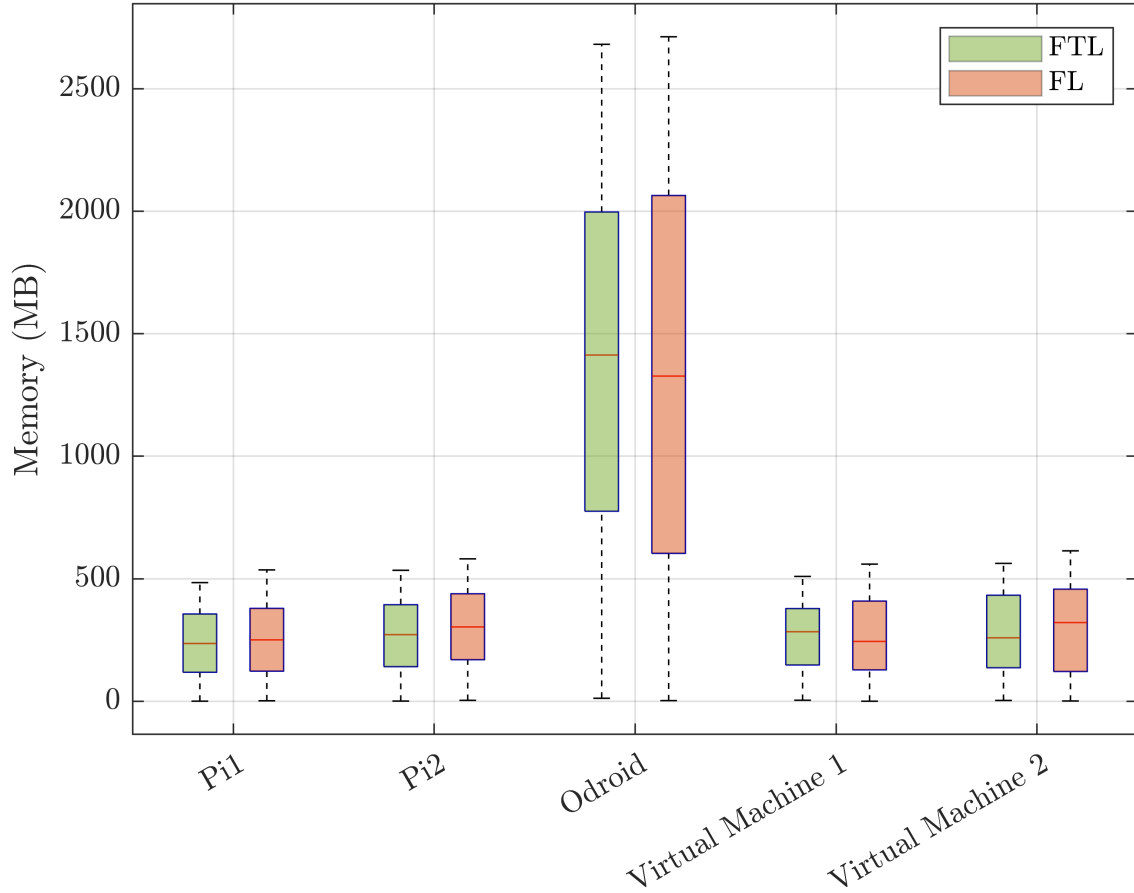


FIGURE 4.29: Memory usage of FTL compared to ordinary FL for 5 clients.

In particular, each client device maximally utilizes approximately half of its available memory when subjected to full-load training of its respective DNN. Specifically, Odroid, with 25% of the dataset allocated to ResNet training, exceeds the halfway mark in memory usage. On the contrary, Raspberry Pis 1 and 2, with data allocations of 10% and 15%, respectively, consume less than half of their total memory. This observed trend in memory utilization aligns proportionally with the assigned dataset volumes, in which Raspberry Pi 2, endowed with a larger dataset compared to Raspberry Pi 1, exhibits increased memory usage.

Further scrutiny of memory usage patterns reveals a nuanced interplay between computational load and processing efficiency. In particular, Raspberry Pi 1 completes

its training tasks before Raspberry Pi 2 in each training round, giving it additional time to recover memory before the next round. Consequently, Raspberry Pi 1 tends to exhibit lower memory consumption relative to Raspberry Pi 2, a phenomenon reflective of the inter-device variability in training durations.

This comprehensive examination elucidates the intricacies of memory management in the context of DL scenarios, shedding light on the nuanced interplay between device specifications, dataset allocations, and training dynamics.

Subsequently, we performed an exhaustive analysis of power consumption across client devices, as illustrated in Figure 4.30. This assessment was facilitated using a digital multimeter capable of dynamically recording and visualizing variations in power consumption over time. In particular, the maximum power consumption corresponds to periods of full-load training, whereas the minimum power consumption occurs during idle intervals when clients wait for the completion of training by other participants.

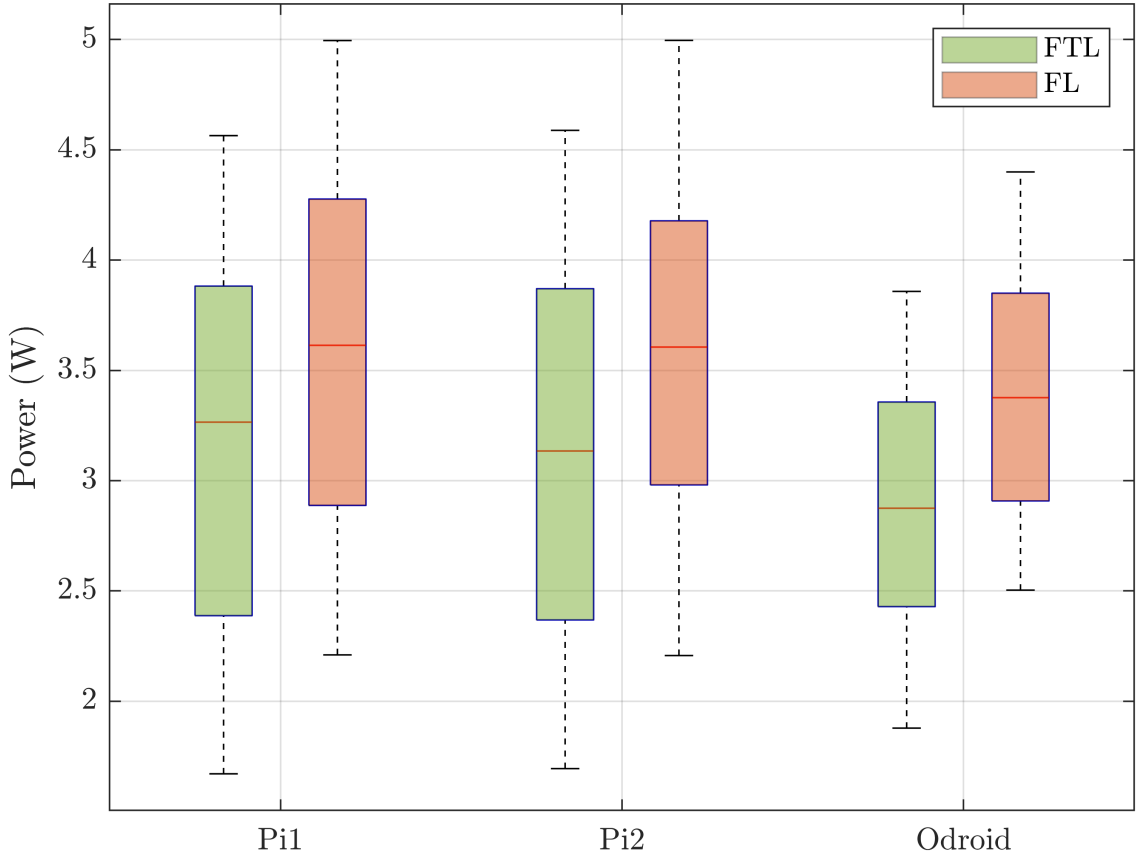


FIGURE 4.30: Power consumption of Raspberry Pis and Odroid when running FTL and FL with 5 clients.

It is worth mentioning that, in Figures 4.28 and 4.30, which depict temperature and power consumption, respectively, the absence of VMs is attributed to the practical limitation of measuring these metrics separately for each VM. In our experimental setup, the VMs, along with the FL server, were implemented on the same CPU, making it challenging to isolate and measure the temperature and power consumption of individual VMs. Consequently, these figures focus solely on the temperature and power consumption of other parts of the system, i.e., Pis and Odroid, without distinguishing between the FL server and VMs.

Finally, we extrapolated these power consumption measurements to derive energy consumption profiles for all devices, as depicted in Figures 4.31–4.33. Intriguingly,

we observed a direct correlation between training effort and energy consumption, whereby intervals of client idleness exhibited a linear increase in energy consumption at a rate equivalent to the minimum power consumption. Conversely, periods of active training were characterized by a steeper increase in energy consumption, correlating with the maximum power consumption.

Furthermore, our analysis revealed a notable difference in total energy consumption between the FTL and FL methodologies. Due to the inherently reduced training effort associated with FTL compared to FL, the total energy consumption incurred by devices utilizing the FTL approach was significantly lower than that observed with FL. This observation underscores yet another advantage of FTL over FL, which is particularly pertinent in scenarios featuring resource-constrained devices, such as Raspberry Pi, and in mobile or vehicular 6G communication scenarios where energy efficiency is paramount. These findings underscore the importance of energy-aware design considerations in optimizing the performance and sustainability of DL systems.

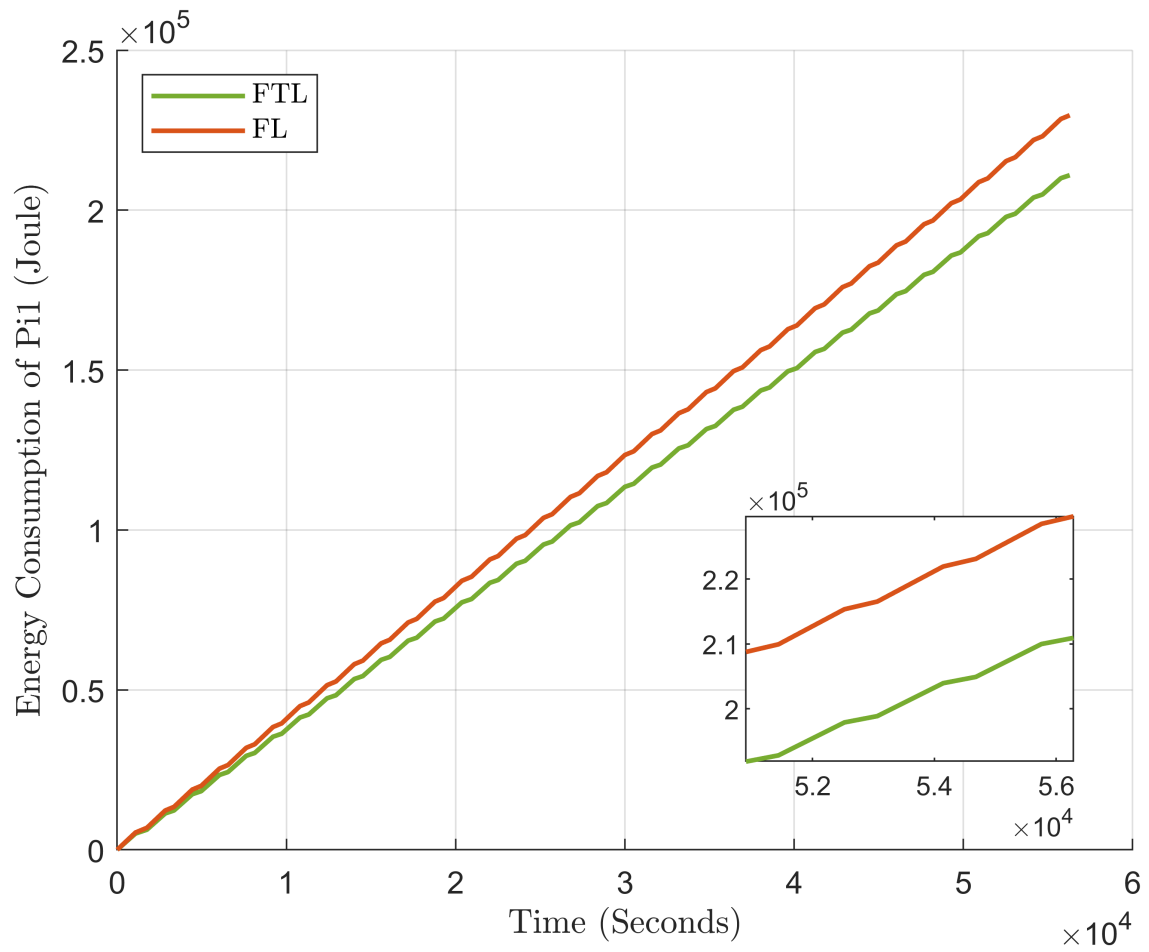


FIGURE 4.31: Energy consumption of Raspberry Pi 1 when running FTL and FL.

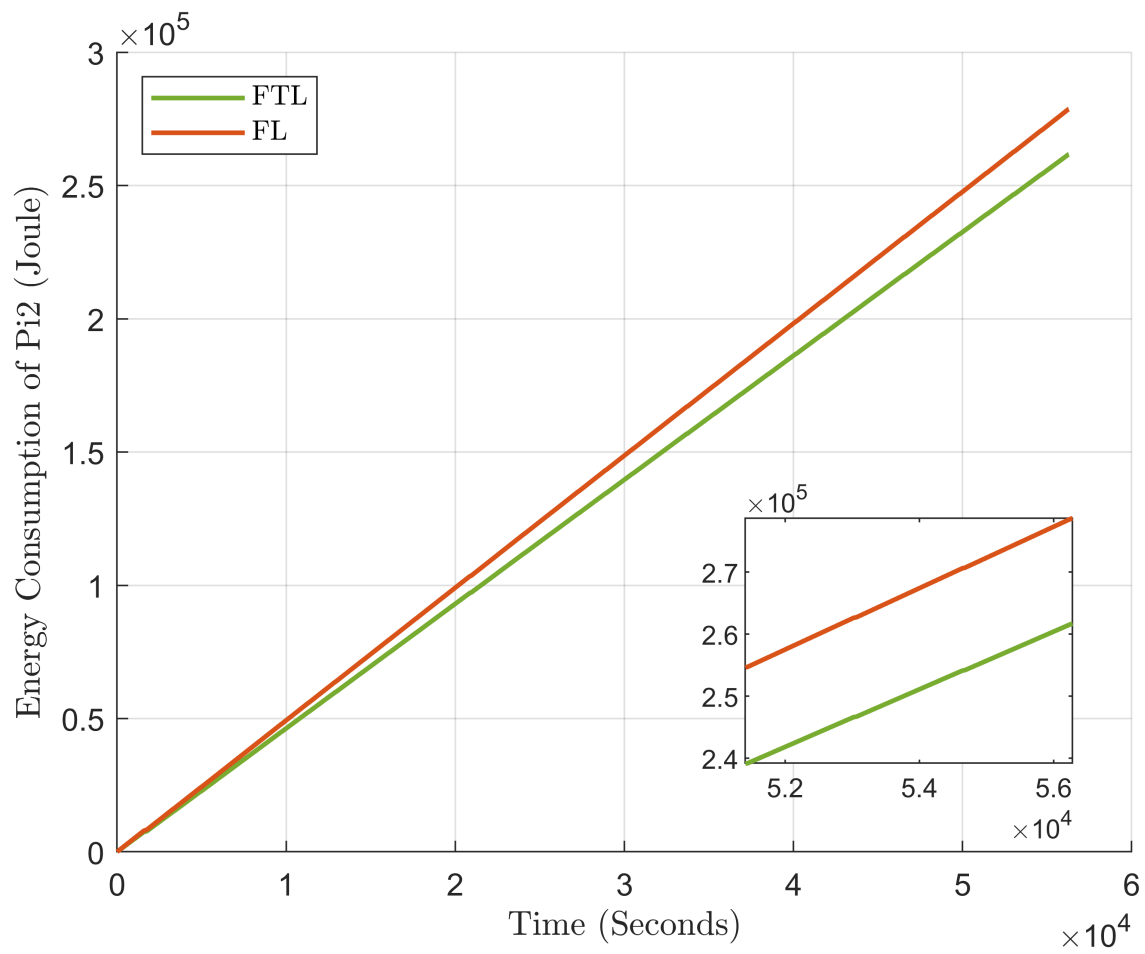


FIGURE 4.32: Energy consumption of Raspberry Pi 2 when running FTL and FL.

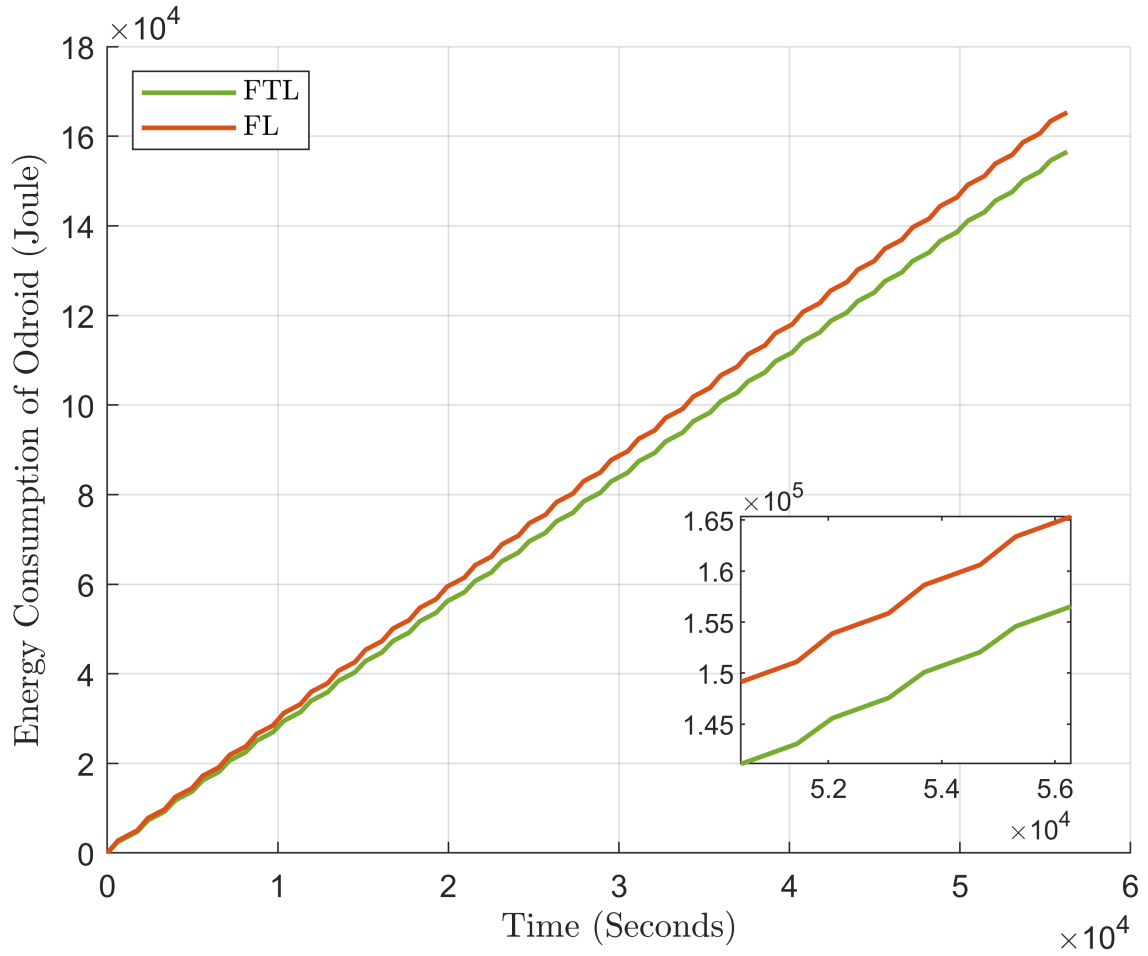


FIGURE 4.33: Energy consumption of Odroid when running FTL and FL.

#### 4.3.6.2 Experiment 2: DL with Three Heterogeneous Clients

DL methods are inherently influenced by factors such as the number of participants and the volume of training data within the dataset. Therefore, it is imperative to explore the impact of these parameters on both the FL and FTL frameworks. Through this investigation, we aim to elucidate the relative sensitivity of FL and FTL to variations in these crucial parameters, highlighting the potential advantages conferred by FTL over conventional FL methodologies.



To this end, we conducted an experiment in which Raspberry Pi 1 and Virtual Client 2, along with their corresponding training datasets, were removed, thus reducing the number of participants to three: one Raspberry Pi, one Odroid, and one virtual machine. Despite this reduction in the user base, the distribution of the data remained consistent with the previous experiment, allocating 15% of the data to the Raspberry Pi, 25% to the Odroid, and 20% to the virtual machine. We experiment with this setting because it mirrors real-world scenarios, such as mobile or vehicular 5G/6G networks, where users may lose connection to the federated server or go out of coverage of the server.

The accuracy and training dynamics measured from these DL approaches were analyzed against the number of training rounds and time, as shown in Figures 4.34 and 4.35, respectively. A comparative analysis with the previous experiment involving five clients (Figures 4.22 and 4.23) revealed interesting information. Despite the reduction in participants, the final accuracy of the FTL approach experienced a slight decrease, from 83% to 81%, while the final accuracy of the FL approach exhibited a more pronounced decrease, from 60% to 55%. This observation underscores the superior robustness of FTL in mitigating the adverse effects of reduced user participation and reduced training data volume, an inherent advantage of FTL over FL.

Furthermore, analysis of the settling time revealed that while the settling time for FTL experienced a modest increase from 9700 to 11,400 s, the FL approach demonstrated a more substantial rise, escalating from 27,000 to 32,100 s. This increased sensitivity of FL to changes in user count and training data volume underscores the inherent resilience of FTL in maintaining stability and performance consistency under varying conditions.

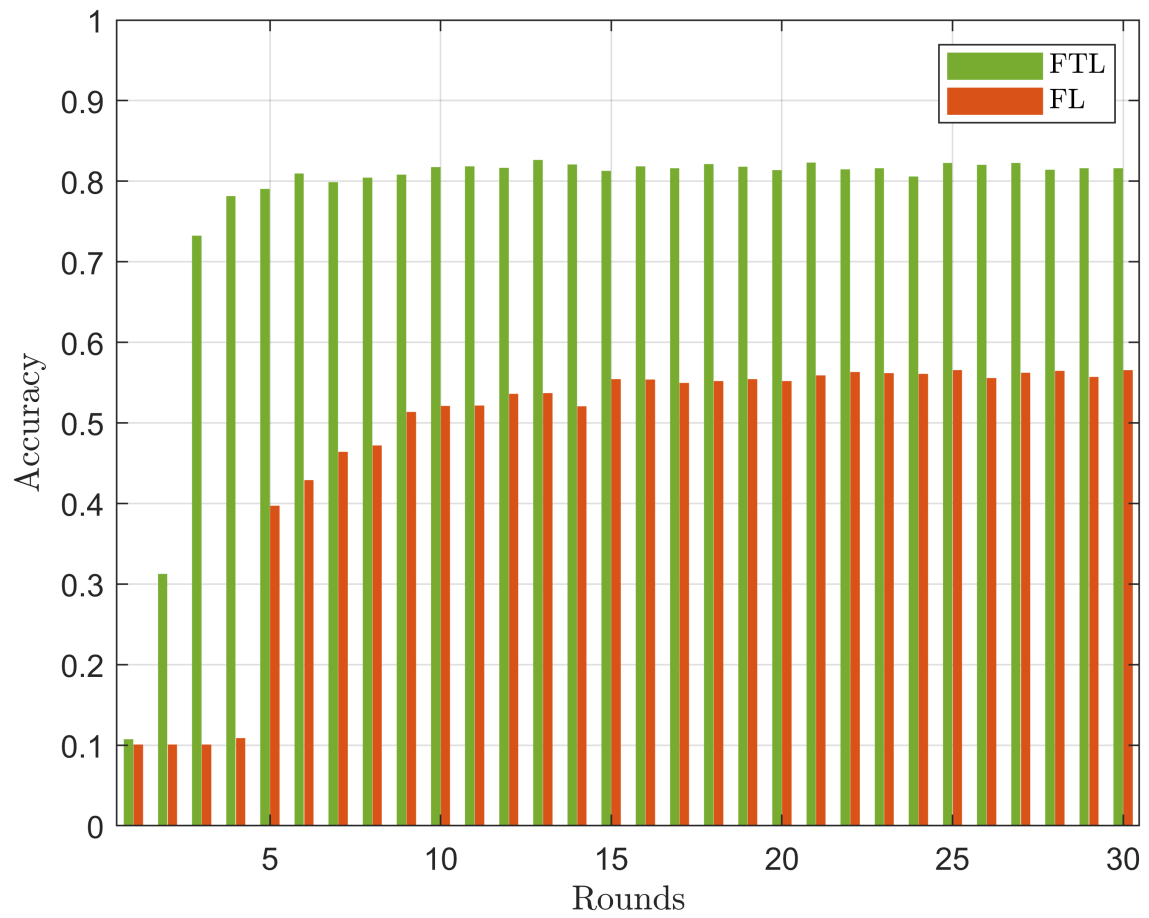


FIGURE 4.34: Accuracy of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. rounds.

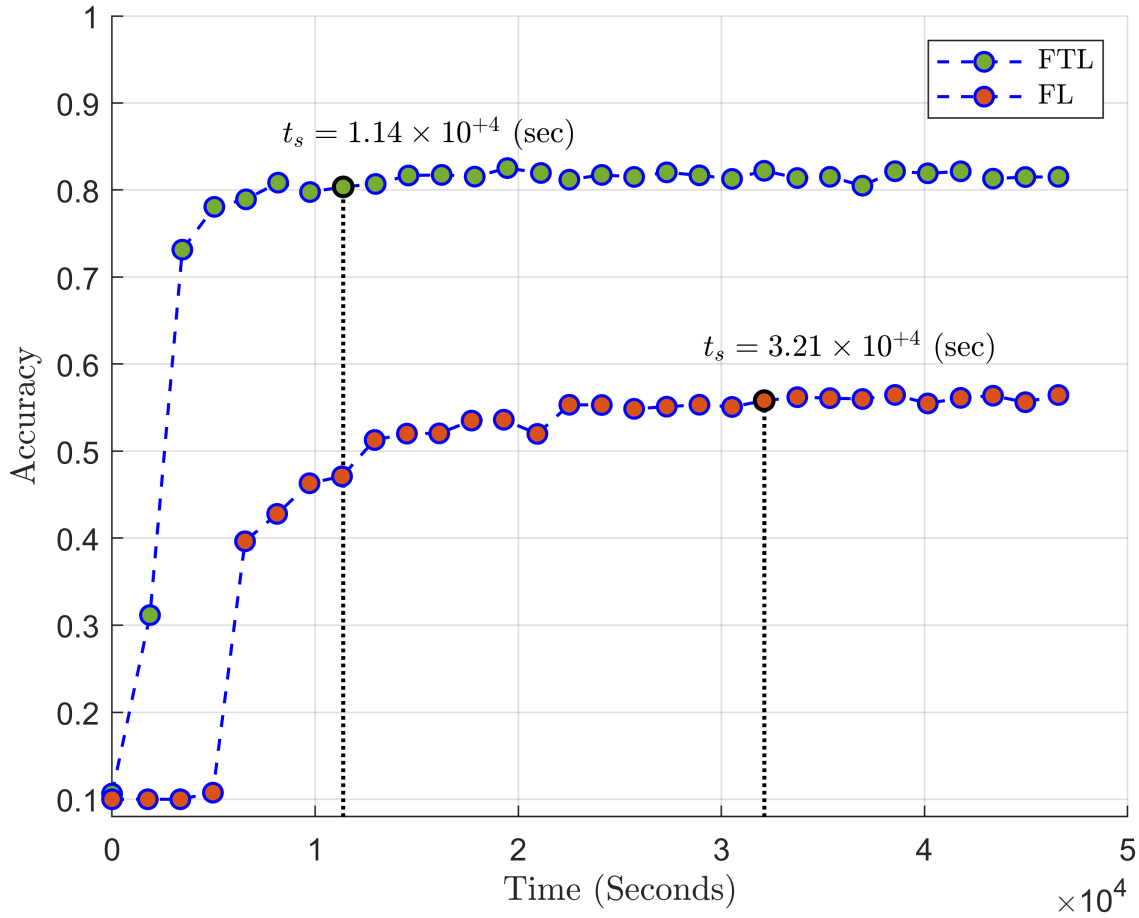


FIGURE 4.35: Accuracy of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. time.

These findings are corroborated by the examination of loss dynamics, in which the categorical cross-entropy loss function is used to compare the performance of FTL and FL across different training rounds and elapsed time, which is demonstrated in Figures 4.36 and 4.37. The superior efficacy of the FTL approach in minimizing the loss function is evident, owing to its utilization of prior training knowledge and avoidance of training from scratch, a distinct advantage of FTL over the FL methodology.

In summary, our comprehensive analysis underscores the robustness and stability of FTL in the face of fluctuating participant counts and training data volumes,

reaffirming its potential as a reliable and efficient framework for DL applications.

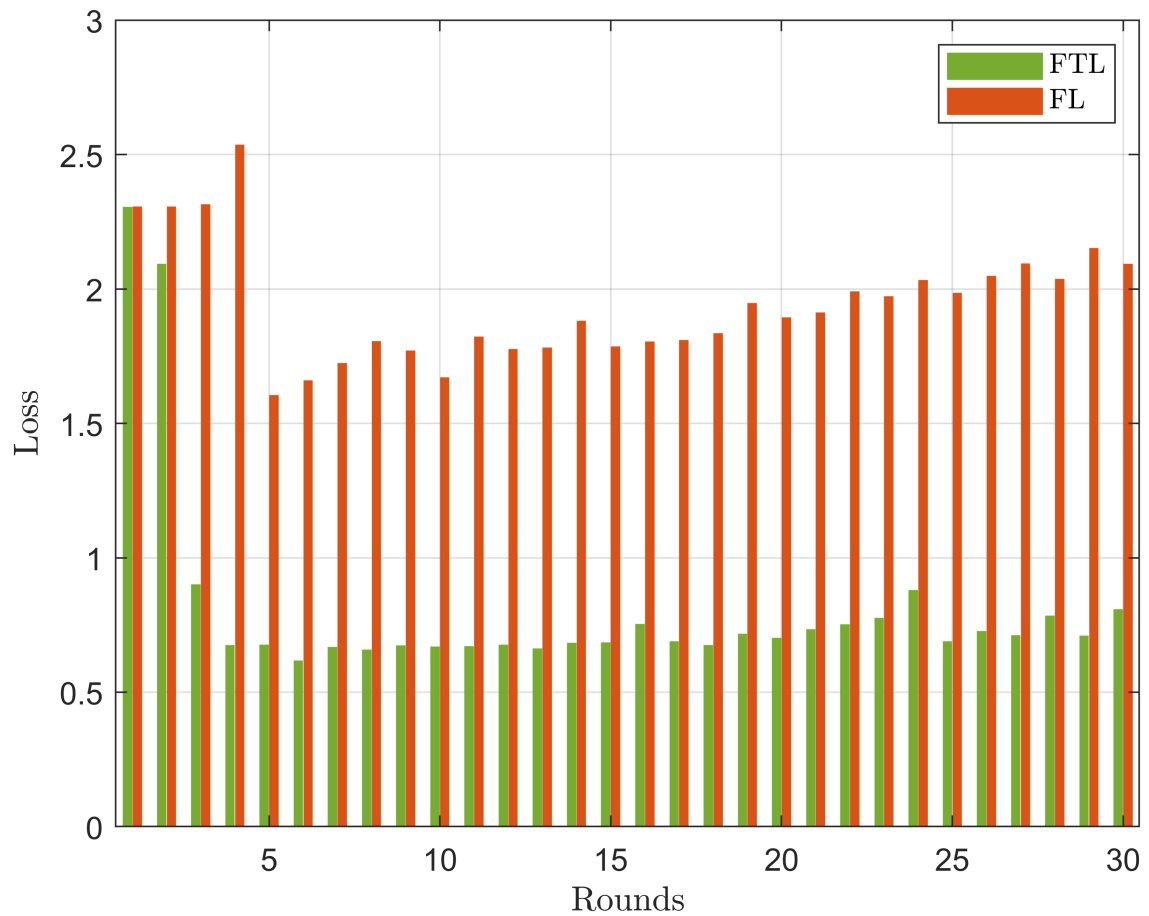


FIGURE 4.36: Loss of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. rounds.

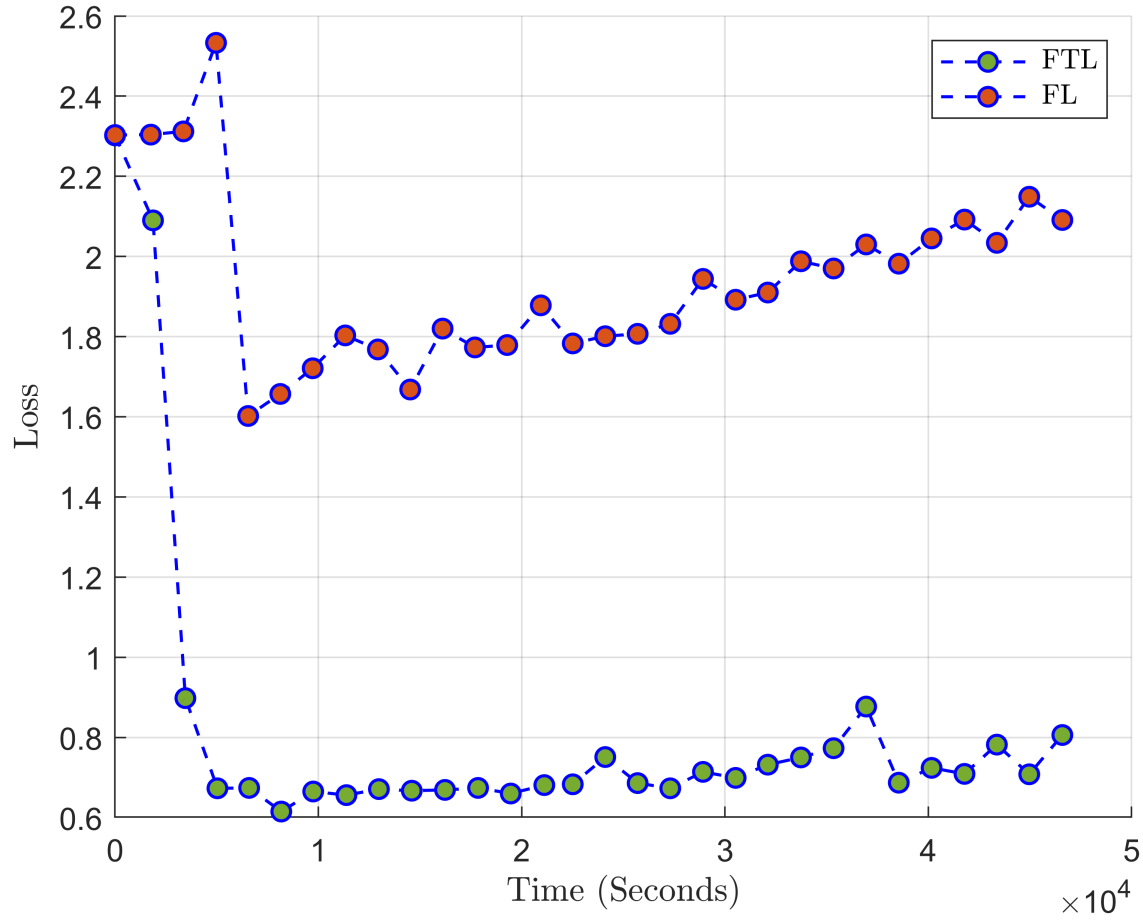


FIGURE 4.37: Loss of FTL for 3 clients, a Raspberry Pi, an Odroid, and a virtual machine, compared to ordinary FL vs. time.

Subsequently, we conducted a comprehensive assessment of key performance metrics, namely load average, temperature, and memory usage, on various client devices during FTL and FL training sessions. The results of this investigation are presented through error bar charts in Figures 4.38–4.40.

Consistent with previous observations, the utilization of FTL manifests in superior performance metrics compared to FL, attributed primarily to the integration of pre-trained networks within the FTL framework. This strategic utilization of pre-existing knowledge mitigates the computational overhead associated with initializing

and training models from scratch, consequently enhancing overall efficiency and effectiveness.

Moreover, our analysis revealed notable disparities in load average and temperature dynamics between the Odroid and Raspberry Pi devices. The Odroid, characterized by superior computational resources relative to the Raspberry Pi, exhibited lower averages and load temperatures. This difference can be attributed to the accelerated training completion times facilitated by the Odroid's enhanced processing capabilities, affording it additional idle time for relaxation and cooling.

Furthermore, an examination of the memory usage patterns revealed that the Odroid device consistently utilized more than half of its available memory, while the Raspberry Pi device consumed less than half of its total memory capacity. This observation underscores the efficient utilization of computational resources across devices, with the Odroid leveraging its enhanced capabilities to support intensive computational tasks, while the Raspberry Pi demonstrates a more conservative approach to memory utilization.

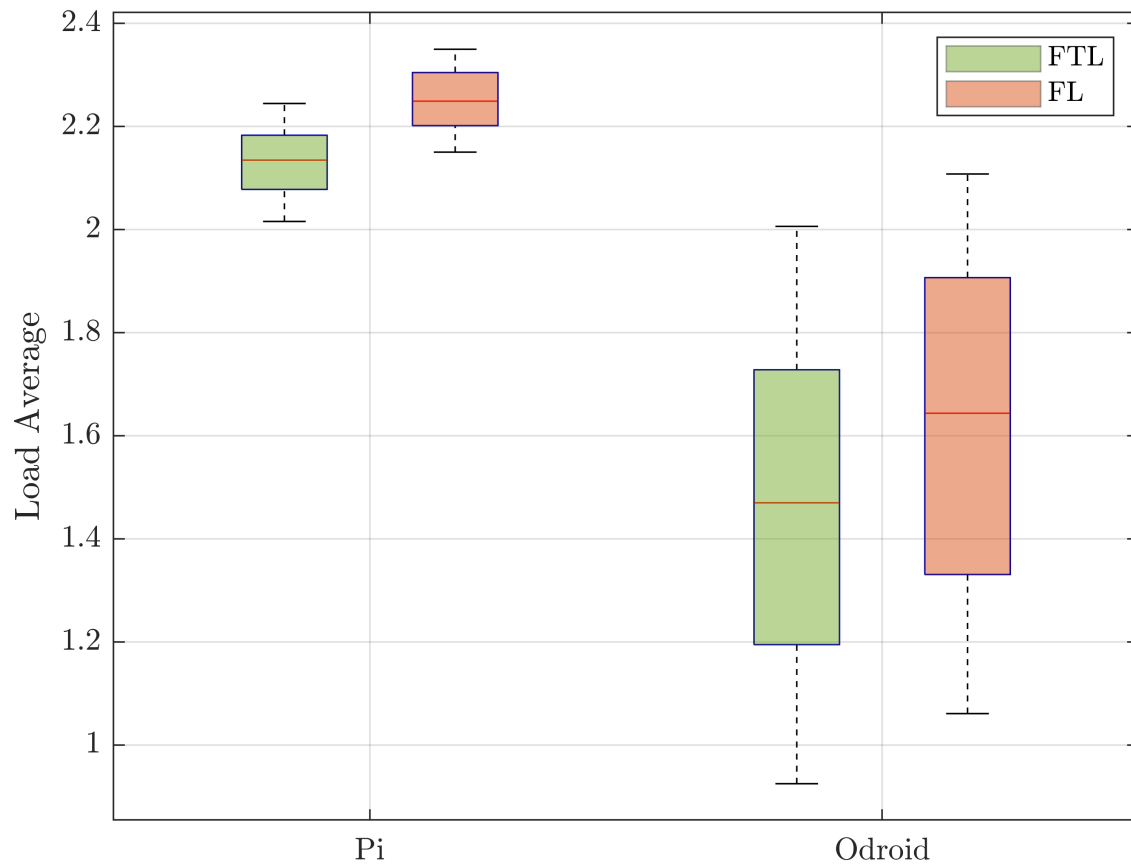


FIGURE 4.38: Average load on the Raspberry Pi and Odroid when running FTL and FL with 3 clients.

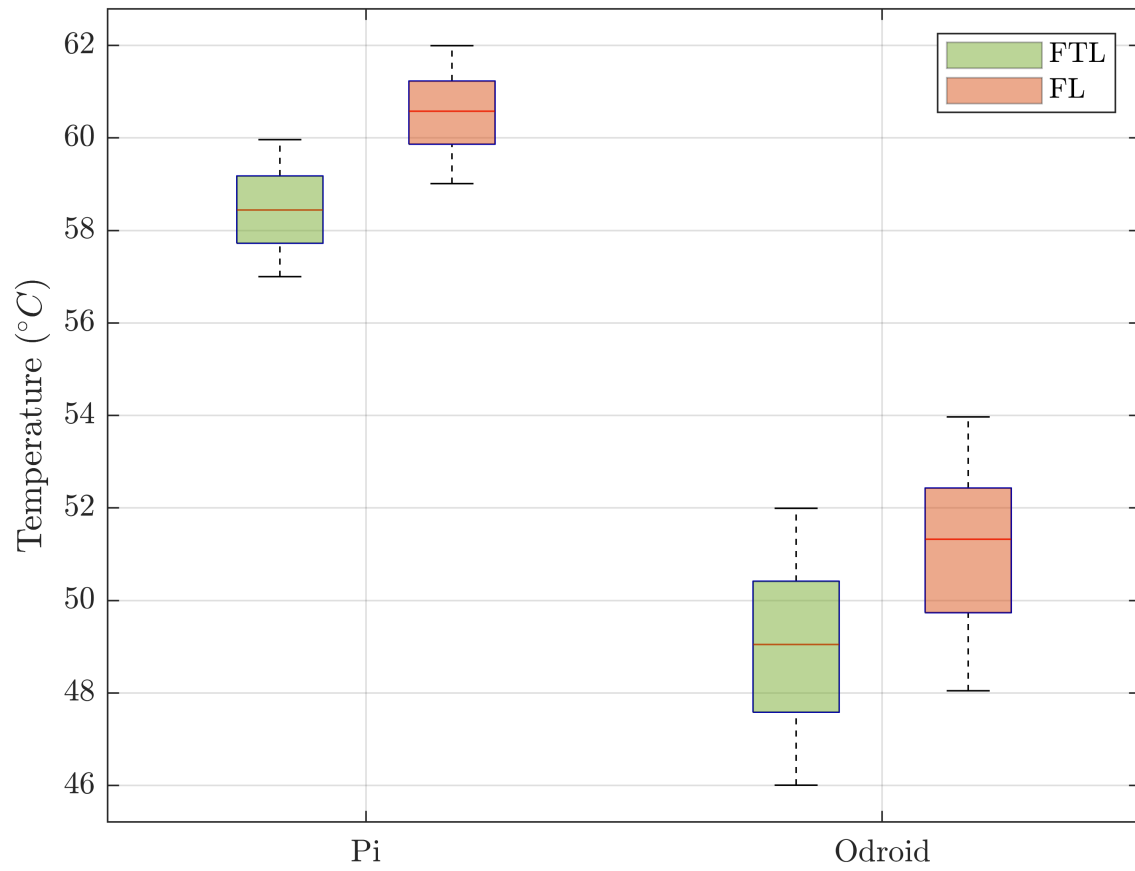


FIGURE 4.39: Raspberry Pi and Odroid temperature when running FTL compared to FL with 3 clients.



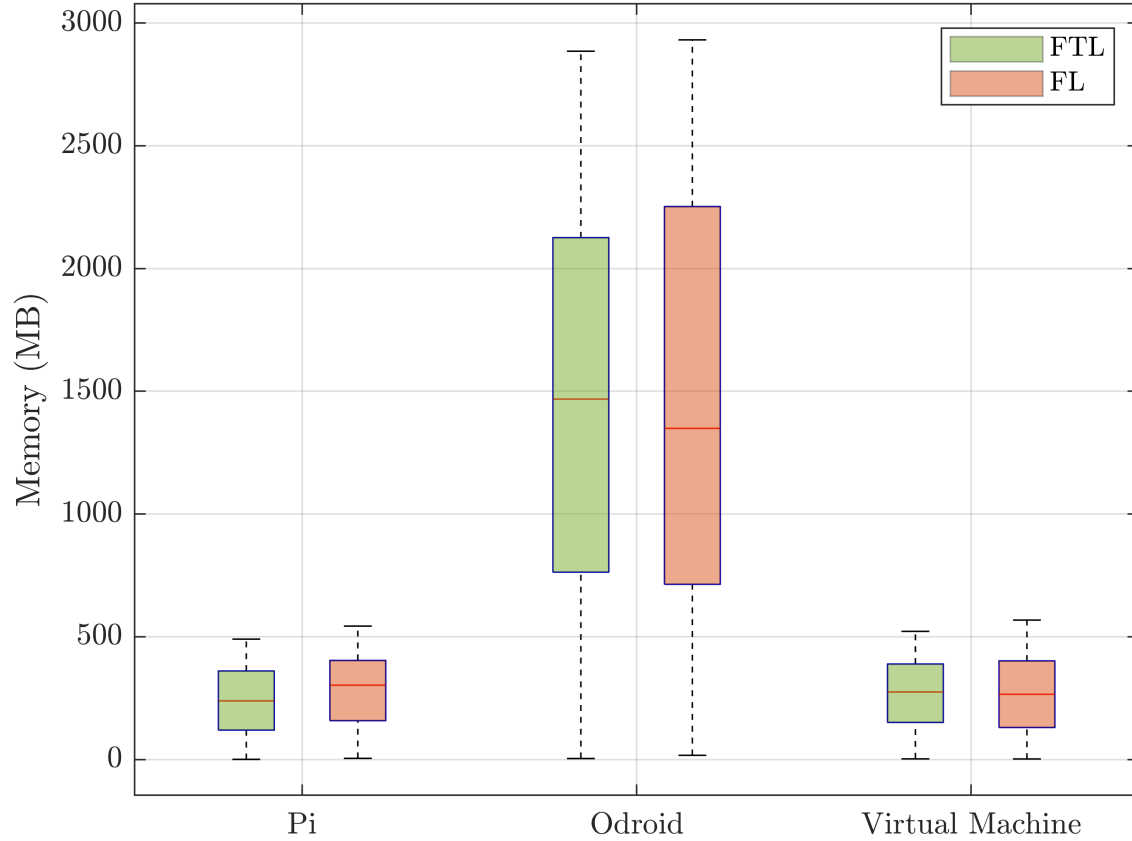


FIGURE 4.40: Memory usage of FTL compared to ordinary FL for 3 clients.

Finally, we meticulously monitored and graphed the power consumption of client devices, as shown in Figure 4.41. Using a digital multimeter with dynamic recording capabilities, we captured fluctuations in power use over time. In particular, peak power consumption coincided with full-load training periods, while minimal power draw occurred during idle intervals as clients waited for the completion of training rounds by other participants.

Following this, we depicted the energy consumption profiles for all devices, as shown in Figures 4.42 and 4.43. We observe a direct correlation between energy consumption and training activity, with idle intervals characterized by a gradual increase in energy consumption at the minimum power rate. On the contrary, the active training phases exhibited a steep rise in energy consumption, in line with maximum power

usage. Furthermore, our analysis revealed that FTL required less training effort compared to FL, resulting in a reduction in overall energy consumption. This underscores FTL's advantages, which are particularly significant in resource-constrained environments such as IoT, mobile, or vehicular 6G communication scenarios, emphasizing its potential to enhance energy efficiency in DL frameworks.

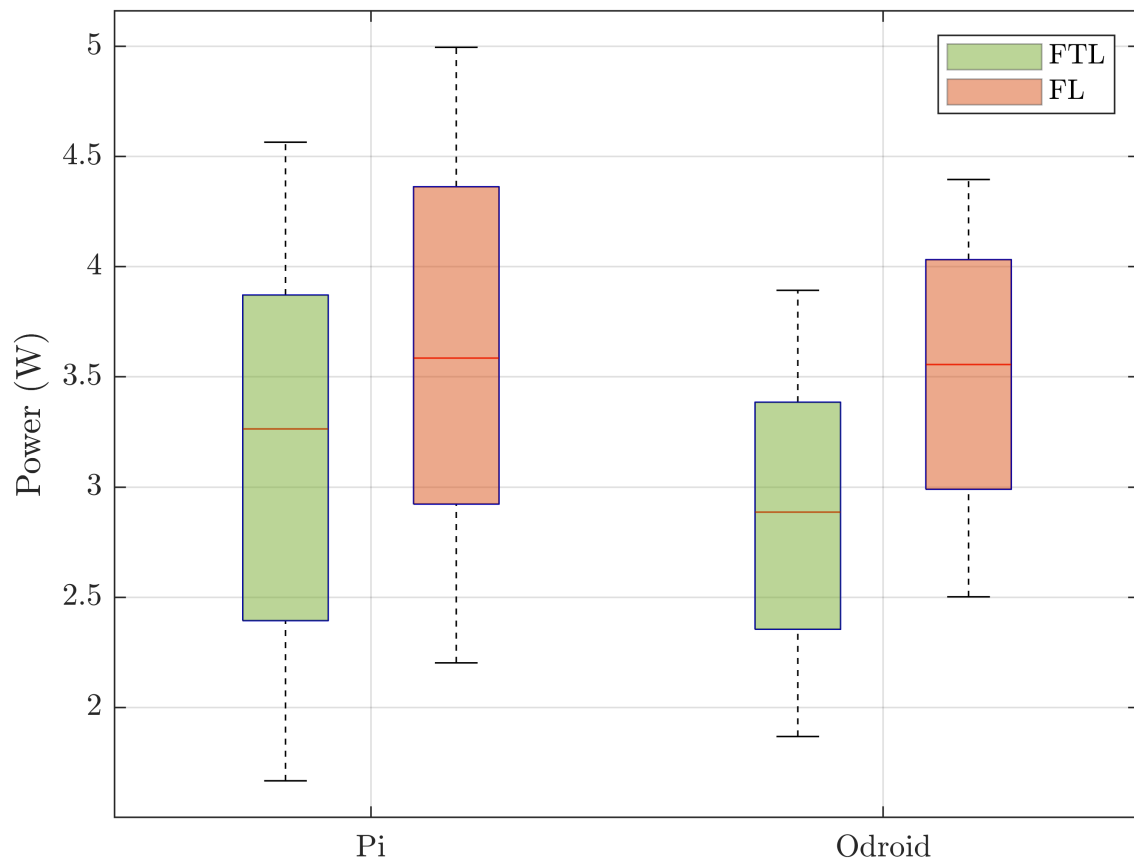


FIGURE 4.41: Power consumption of Raspberry Pi and Odroid when running FTL and FL with 3 clients.

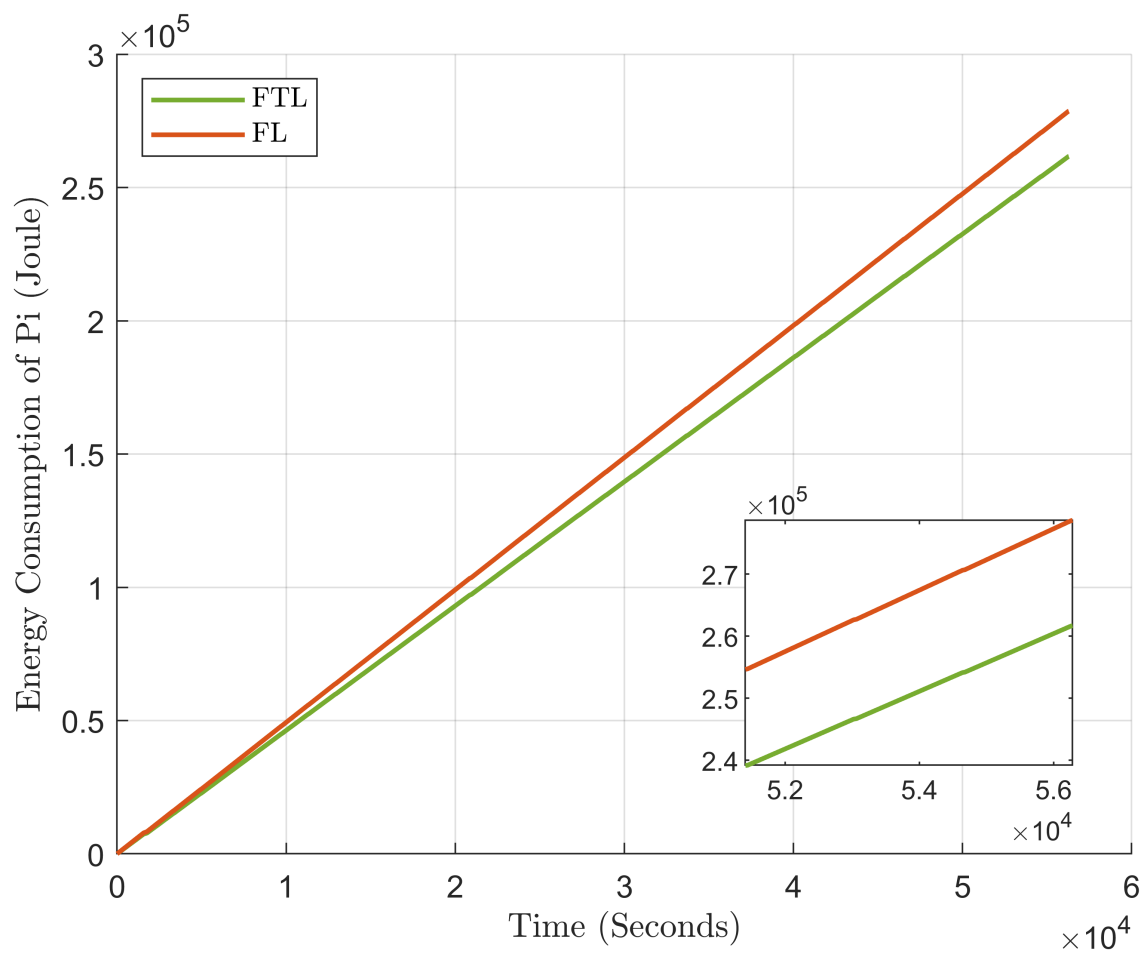


FIGURE 4.42: Energy consumption of Raspberry Pi when running FTL and FL.

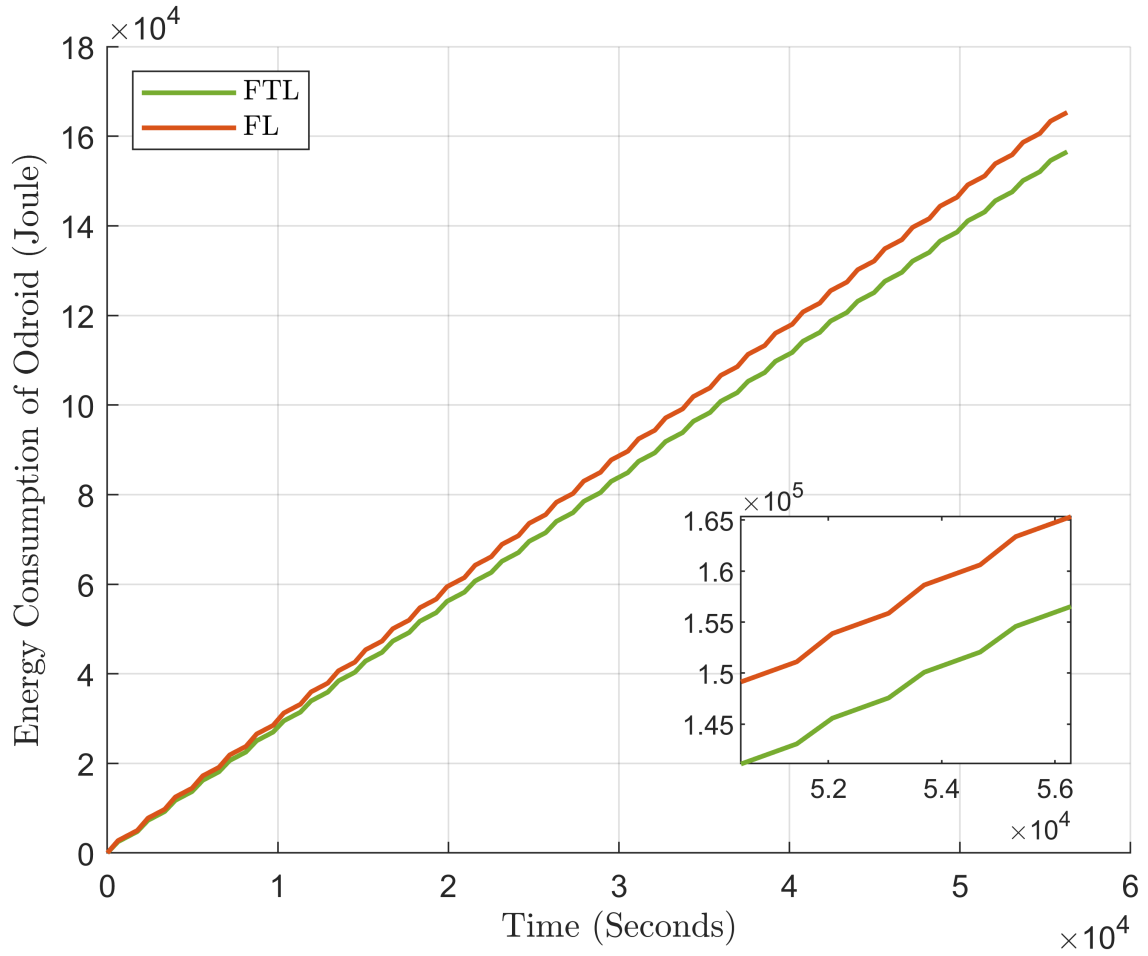


FIGURE 4.43: Energy consumption of Odroid when running FTL and FL.

### 4.3.7 Conclusion

In conclusion, this study conducted a comprehensive exploration of DL frameworks in response to the evolving landscape of 6G technology and the imperative for a fully connected distributed intelligence network for IoT devices. Recognizing the inherent challenges posed by the heterogeneous nature of clients and data, we embarked on an investigation of FTL as an alternative to traditional FL techniques. Through meticulous design, implementation, and evaluation, we elucidated the efficacy and superiority of FTL, particularly tailored to the diverse constraints of IoT platforms.

Our findings underscore the notable performance advantages offered by FTL over FL, particularly characterized by faster training and higher accuracy, facilitated by the incorporation of TL methodologies. Real-world measurements further substantiated these assertions, revealing improved resource efficiency with lower average load, memory usage, temperature, power, and energy consumption in FTL implementations compared to their FL counterparts.

Moreover, our experiments unveiled FTL's resilience in scenarios characterized by sporadic client participation, where users depart from the server's communication coverage, resulting in a reduced number of clients and training data availability. This adaptability underscores FTL's effectiveness in environments with limited data, clients, and resources, offering valuable insights into the convergence of EC and DL for 6G IoT applications.

By addressing practical challenges through innovative implementation strategies, such as memory swap functionality and asymmetric data distribution, we have advanced the understanding and applicability of DL frameworks in resource-constrained environments. Looking ahead, future research endeavors could further explore the scalability and optimization of FTL methodologies, fostering the continued evolution of distributed intelligence networks in the era of 6G technology.

## Chapter 5

# Deep Reinforcement Learning for Dynamic Adaptive Streaming

Some content of this chapter is based on the following articles [168]:

1) **David Naseh**, Arash Bozorgchenani, and Daniele Tarchi. "Deep Reinforcement Learning for Edge-DASH-Based Dynamic Video Streaming." In *2025 IEEE Wireless Communications and Networking Conference (WCNC)*, pp. 1-6. IEEE, 2025.

2) Arash Bozorgchenani, **David Naseh**, Daniele Tarchi, Sergio A. Salinas Monroy, Farshad Mashhadi, Qiang Ni, "Reinforcing Edge-DASH: Deep Reinforcement Learning for Multi-Objective Streaming Optimization." Submitted to the *IEEE Transactions on Mobile Computing* (under revision).

### 5.1 Introduction

By 2025, mobile video streaming is expected to account for 76% of mobile data traffic [169]. With mobile subscriptions projected to reach 8.4 billion by 2029 and

5G networks covering up to 65% of the global population, ensuring QoE for users has become a key research focus. The adoption of 5G is further driving mobile data traffic, enabling more immersive media formats. The diversity in user demands, influenced by network conditions and device capabilities, presents challenges for video service providers in maintaining optimal QoE. Adaptive Bit Rate (ABR) streaming and Dynamic Adaptive Streaming over HTTP (DASH) have emerged as solutions [170], allowing users to stream video in resolutions that suit their data rate and preferences. However, a network-only strategy is inadequate, as rate adaptation must consider user preferences and device factors like screen size and bandwidth [171]. While cloud computing supports DASH services, Cloud-DASH has drawbacks, such as high latency and core network congestion [172]. MEC mitigates these issues by providing computation and storage resources closer to users at the network edge [173, 174]. MEC helps meet 5G's low-latency requirements, but an efficient User-to-Server Allocation (USA) mechanism is necessary to balance traffic across cloud and edge resources.

A cache hit occurs when a video chunk with the requested resolution is available, which makes ABR-aware edge caching more complex. In ABR streaming, having simply a video chunk in the cache is insufficient; the chunk must be stored at the required bit rate [175]. To address this issue, given the limited storage at the edge, we propose incorporating transcoding functionality at the Base Stations (BSs), which improves performance by eliminating the need to cache all possible bitrate levels. However, real-time video transcoding is highly computationally demanding, and transcoding multiple videos simultaneously can quickly deplete the processing capacity of MEC servers. Therefore, it is crucial to develop a bitrate delivery strategy that optimizes the use of processing resources.

Several studies have addressed the Bitrate Allocation (BrA) problem. Mehrabi et

al. [176] proposed a greedy-based scheduling algorithm that periodically solves the USA and BrA problems, aiming to balance the load while considering various QoE metrics. However, their approach is entirely network-driven, overlooking client-side limitations and focusing on maximizing bitrate for all users. Bayhan et al. [177] studied BrA in WiFi Access Points to facilitate cache delivery. Their proactive approach considered a tolerable difference between the requested and delivered bitrates, using a compositional Pareto-algebraic heuristic. Although their model has some similarities with ours, they did not incorporate transcoding techniques, which play a crucial role in enhancing QoE. Some works have also considered hybrid edge-cloud architectures. Tao et al. [178] proposed an edge-cloud-assisted predictive adaptive streaming framework for mobile networks with unreliable data rate predictions, using slow fading to optimize long-term scheduling risk. Yan et al. [179] developed a hybrid edge-cloud framework to optimize client rate adaptation in cellular networks.

In this work, we jointly address the problems of BrA and USA. First, we model the system and formulate a joint BrA-USA optimization problem. Then, we propose a DRL approach to solve this problem, determining the optimal streaming source for users, which can be from the edge, the Macro layer, or the cloud. Our solution ensures that users receive the most suitable bitrate while maintaining QoE by addressing the BrA problem. The main contributions of our work are as follows:

1. We introduce a reactive streaming strategy within a 4-tier network topology, where users can stream from either the edge (small cells), a Macro cell with wider coverage, or the cloud. Additionally, we incorporate transcoding capabilities in the edge layer,
2. We formulate a joint BrA-USA problem to optimize bitrate delivery in a reactive, multi-layer network environment,



3. We develop a DDPG method, which combines DRL and off-policy deterministic policy gradient approaches, to solve this joint problem. To the best of our knowledge, our approach is novel in this area.

### 5.1.1 System Model and Problem Formulation

#### 5.1.2 System Model

We consider a macro-cellular network that covers a specific area, where a Macro Base Station (MBS) connects to the cloud via high-capacity links. The MBS contains a Macro Edge Server (MES), and within the Macro cell,  $S$  Small-cell Base Stations (SBSs) are co-located with Small-cell Edge Servers (SESs). The network provides video streaming services to Edge Clients (ECs), who request videos in varying resolutions. SESs store video content at multiple bitrates, whereas the MES holds the highest resolution versions.

Since ECs may prefer lower bitrates due to device constraints (e.g., battery life or data limits), our system adopts a reactive approach that takes user requests into account to ensure satisfactory QoE. ECs, represented by  $N$  users, connect to the closest base station, each requesting chunks from a video catalog, each chunk encoded at various bitrates. SESs can transcode video chunks to lower bitrates on requests, while the requested bitrate may differ from the delivered one.

There are  $N$  ECs scattered throughout the area, represented by indices  $\mathcal{U} = \{u_1, u_2, \dots, u_N\}$ . Each EC connects to the nearest BS according to signal strength. The available videos are represented by  $\mathcal{V} = \{v_1, v_2, \dots, v_M\}$ , and each video is divided into  $K$  chunks, each encoded at multiple bitrates. The SESs are capable of transcoding chunks to lower bitrates. ECs can request chunks from a range of

bitrate levels, denoted as  $\mathcal{O} = \{o_{\min}, o_{\max}\}$ , and the bitrate of the  $k$ th chunk of the  $m$ th video is denoted as  $o_{m,k}$ . Each chunk has a fixed duration, typically between 2 and 10 seconds.

ECs' bitrate requests are denoted as  $r_i^{o_{m,k}}(t)$ , with the general notation  $r_i(t)$  used for simplicity. The set of all EC requests is  $\mathcal{R}(t) = \{r_1(t), \dots, r_i(t), \dots, r_N(t)\}$ . The actual delivered content is represented by  $\hat{\mathcal{R}}(t) = [\hat{r}_i(t)]_{i=1}^N$ , where the delivered bitrate  $\hat{r}_i(t)$  may differ from the requested bitrate.

The SESs provide caching and transcoding capabilities. Each SES has limited capacity, which allows it to store a certain number of videos and chunks out of the total of  $M$  videos and  $K$  chunks. Due to capacity constraints, not all bitrates can be cached, so each chunk is stored at a specific bitrate if cached at all.

At any given time  $t$ , an EC may request a video chunk at a certain bitrate. The requested chunk may be delivered from an SES, MES, or cloud, depending on availability. To maximize QoE, we prioritize delivering the requested chunk from the SES. If the SES cannot meet the request, the EC can go back to the MES or the cloud. The MES stores all videos at the highest bitrate, whereas the cloud stores all videos at all bitrates. The goal is to optimize the delivered bitrate, matching it as closely as possible to the requested bitrate while meeting QoE requirements. The following options describe the video delivery process:

1. Direct Edge hit: The requested chunk is cached at the SES in the requested bitrate.
2. Transcoding Edge hit: The chunk is cached at a higher bitrate, but is transcoded to the requested lower bitrate.

3. MES hit: The EC may access the MES if the chunk is either not cached at the SES or cached at a lower bitrate, or if MES provides better QoE or lower cost as MES stores all videos only with the highest bitrate.
4. Streaming from the cloud: If none of the above options is feasible, or if the cloud offers better QoE or lower cost, the EC streams from the cloud. This is an option; however, interaction with the cloud is preferred to be limited due to the traffic burden on the backhaul.

The ability of SESs to transcode is restricted by a shortage of computational resources. Let  $\eta_o^s$  represent the computing resources required for transcoding a chunk at a certain cached bitrate to a lower bitrate  $o$ . Transcoding to lower bitrates consumes more resources, i.e.,  $\eta_o^s < \eta_{o^-}^s$  if  $o^- < o$ . The indicator function  $\mathbb{R}_i^o$  is equal to 1 if the request of EC  $i$  requires transcoding to bitrate  $o$ . The total computing resource constraint on SESs for transcoding is defined as:

$$\sum_{i=1}^{N_s} \sum_{o=1}^O \eta_o^s \cdot \mathbb{R}_i^o \leq \Omega^s \quad \forall s \quad (5.1)$$

where  $\Omega^s$  is the maximum available computing resource for transcoding at an SES and  $N_s$  is the number of ECs covered by an SES. We assume that the cloud does not have any limitation in computation capacity. The cost of transcoding a chunk is measured by CPU usage on the cache servers.

### 5.1.3 Problem Formulation

Our objective is to maximize EC satisfaction by delivering the appropriate bitrate using optimal BrA and USA mechanisms, while considering edge resource constraints.

Since each EC has multiple streaming sources (SBS, MBS, or cloud), the USA decision is made independently for each request. A key metric for effective BrA is avoiding stalling during streaming, which directly impacts QoE. However, USA is also tied to BrA since the choice of streaming source can depend on whether the requested chunk needs transcoding. Given the limited edge resources, all streaming options must be included in the optimization of bitrate and server allocation. In some cases, delivering a lower bitrate than requested may be preferable to prevent stalling and meet the USA requirements. Thus, the optimal USA decision may sometimes come at the cost of lower bitrate delivery. Both BrA and USA decisions are also influenced by the content cached in SESs. An effective caching strategy can improve streaming by increasing direct edge hits or reducing transcoding resource usage. However, due to the cache capacity limitations of each SES, only a limited number of chunks at specific bitrates can be stored. Since this work does not focus on caching, we assume a random caching strategy at the edge.

To define the cost functions, we use the *Weber–Fechner* law, which explains the relationship between the actual changes in stimuli and human perception. This law has been shown to model user satisfaction in communication systems and multimedia applications, particularly following a logarithmic relationship for QoE [180, 181].

In the USA problem, each EC is assigned to a single server (SES, MES, or cloud) for streaming. Let  $a_{i,j}$  be a variable that denotes whether EC  $u_i$  streams from the  $j$ th node out of SES, MES, or the cloud. The USA matrix  $A \in \mathbb{R}^{N \times 3}$  shows the allocation for all ECs, with  $\sum_{j=1}^3 a_{i,j} = 1$  for each EC. Given the reactive approach to BrA, where ECs are served based on their requested bitrates, and the joint optimization of BrA and USA, the joint cost function for each EC request is defined as:

$$\Upsilon_{\phi_k^m}(\hat{r}_i(t), r_i(t)) = \alpha \log \frac{\max(\hat{r}_i(t), r_i(t))}{\min(\hat{r}_i(t), r_i(t))} \quad (5.2)$$

where  $\alpha$  is a positive constant that reflects the significance of the cost, and  $\hat{r}_i(t)$  is dependent on the value of  $a_{i,j}$  as it identifies the streaming source, which is why it is a joint problem. The difference between requested and delivered bitrates at lower levels has a more significant impact on the cost compared to higher levels. This encourages the algorithm to allocate losses, if unavoidable, to higher bitrate levels, where user dissatisfaction will be less pronounced. Additionally, as the gap between the requested and delivered bitrates grows, so does the cost. Therefore, when  $\hat{r}_i(t) = r_i(t)$ , the cost is zero. We define the total joint BrA-USA cost function for all ECs:

$$\hat{Y}(\hat{\mathcal{R}}(t), \mathcal{R}(t)) = \sum_{i=1}^N \Upsilon_{\phi_k^m}(\hat{r}_i(t), r_i(t)) \quad (5.3)$$

We define the optimization problem as

$$\mathbf{P1} : \underset{\hat{\mathcal{R}}, A}{\text{minimize}} \left\{ \sum_{t=1}^T \sum_{i=1}^N \left( \Upsilon_{\phi_k^m}(\hat{r}_i(t), r_i(t)) \right) \right\} \quad (5.4)$$

subject to

$$\mathbf{C1} : \text{Eq. (5.1)} \quad \forall s \quad (5.5)$$

$$\mathbf{C2} : \sum_{j=1}^3 a_{i,j} = 1, \quad \forall u_i \in \mathcal{U} \quad (5.6)$$

In (5.4), the goal is to minimize the difference between the requested bitrate and the delivered bitrate for all ECs over the time horizon  $T$ . The optimization focuses on the delivered bitrate vector  $\hat{\mathcal{R}}$  and the server assigned for streaming,  $A$ . The transcoding computing constraint for each SES is represented in (5.5), while the USA condition, ensuring that each EC streams from either the SES, MES, or cloud, is shown in (5.6).

## 5.2 Proposed Solution

### 5.2.1 Preliminaries

We assume that each EC is associated with the SES that provides the highest SINR. In each time slot, some ECs make a request, while others are in playback mode, having made previous requests. Let  $\mathcal{U}(t)$  represent the ECs connected to SES with no request and  $\mathcal{U}(t)$  those with new requests. The total number of ECs in slot  $t$  is given by  $|\mathcal{U}(t)| + |\mathcal{U}(t)| = N$ .

We design a DRL-based algorithm located in the MES that generates BrA decisions for the ECs. Each small cell includes an environment  $E$ , states  $\mathcal{S}$ , and actions  $\mathcal{A}$ , with a reward function  $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ . At each step  $t$ , the SES observes the state  $s_t \in \mathcal{S}$ , selects an action  $a_t \in \mathcal{A}$  using policy  $\pi$ , and receives a scalar reward  $r_t = r(s_t, a_t) \in \mathcal{R}$  proportional to the QoE. The agent transitions to the next state  $s_{t+1} \in \mathcal{S}$  with probability  $p(s_{t+1}|s_t, a_t)$ . The actor's objective is to find the optimal policy  $\pi^*$  that maximizes the long-term expected reward:

$$R_t = \sum_{i=t}^T \gamma^{(i-t)} \cdot r(s_i, a_i), \quad (5.7)$$

where  $\gamma \in [0, 1]$  is the discount factor.

The DRL framework, based on the Wolpertinger Policy [182], includes three components:

1. **The actor:** The actor network finds a proto-action  $\hat{a} \in \mathcal{A}$  based on the decision policy, updated after each step. The actor is parameterized by  $\theta^\mu$  and maps states  $\mathcal{S}$  to actions  $\mathcal{A}$ , providing a proto-action  $\hat{a}$  for the current state  $\mu(s|\theta^\mu) = \hat{a}$ .

2. ***K-Nearest Neighbors (KNN)***: The KNN maps the proto-action  $\hat{a}$  to a set of valid actions  $\mathcal{A}_k$  to simplify action selection in large spaces:

$$\mathcal{A}_k = g_k(\hat{a}_t), \quad (5.8)$$

where

$$g_k = \underset{a \in \mathcal{A}}{\operatorname{argmin}}^k |a - \hat{a}|_2. \quad (5.9)$$

$g_k$  is a  $k$ -nearest-neighbor mapping from a continuous space to a discrete set, and it returns the  $k$  actions in  $\mathcal{A}$  that are closest to  $\hat{a}$  by  $L_2$  distance, i.e.,  $|a - \hat{a}|_2$ .

3. ***The critic***: The critic evaluates the expanded actions from KNN and selects the one with the highest Q-value:

$$a_t = \arg \max_{a_j \in \mathcal{A}_k} Q(s_t, a_j). \quad (5.10)$$

The deterministic target policy for the critic is:

$$Q(s_t, a_j | \theta^Q) = \mathbb{E}_{r_t, s_{t+1}} [r(s_t, a_j) + \gamma Q(s_{t+1}, a_{t+1} | \theta^Q)], \quad (5.11)$$

where  $\theta^Q$  are the critic network parameters. The action  $a_t$  that maximizes the Q-value is:

$$a_t = \arg \max_{a_j \in \mathcal{A}_k} Q(s_t, a_j | \theta^Q). \quad (5.12)$$

### 5.2.2 DRL-based Solution for BrA-USA

We introduce a DRL-based method located in the MES to solve **P1**. Using the DDPG algorithm, the MES learns a dynamic BrA policy, selecting bitrate actions for ECs based on the observed environment states. The agent has no prior knowledge of the environment, which means it does not know the number of ECs, or bitrate demand, making the learning process model-free. The critic network  $V(x)$  and the actor  $\pi_{\theta_i}(o)$  are parameterized by  $\theta = \{\theta_c, \theta_s\}$ .

The DRL takes the demand profiles from  $N$  ECs,  $\mathcal{R}(t) = \{r_1(t), \dots, r_N(t)\}$ , and outputs the BrA decision vector  $\hat{\mathcal{R}}(t) = \{\hat{r}_1(t), \dots, \hat{r}_N(t)\}$ . Only  $|\mathcal{U}(t)|$  ECs have new demand in slot  $t$ , while the others have an entry of 0. The DRL-based BrA-USA DDPG is defined as follows:

**State Space:** Agent's state is determined by the full system observation includes ECs' buffer length ( $\mathbf{B}(t)$ ), and transcoding capacity at  $t$ , i.e. the agent's state is  $s_t = [\mathbf{B}(t), \bar{\Omega}]$ .

**Action Space:** Agent finds an action matrix  $\bar{\mathcal{R}}(t) \in \mathbb{R}^{N \times \nu}$ , where  $\nu = 2 + |\mathcal{O}|$ , representing all streaming options from SES (considering all transcoding options) plus the two options of MES and Cloud. The matrix is converted to vector  $\hat{\mathcal{R}}(t) \in \mathbb{R}^{1 \times N}$  by assigning the highest probable bitrate using Softmax. The action space becomes  $(N)^\nu$ , and is continuous, optimized by DDPG.

**Reward Function:** The agent aims to minimize **P1** while meeting the EC constraints. The reward function  $r_t$  accounts for the allocated bitrate and the penalty on transcoding for all SESs:

$$r_t = - \left( w_1 \cdot \sum_{i=1}^N \Upsilon_{\phi_k^m}(\hat{r}_i(t), r_i(t)) \right)$$



$$+ w_2 \cdot \sum_{\forall s \in \mathcal{S}} p_s^{\text{trans}} \left( \hat{\mathcal{R}}^s(t) \right) \Big) \quad (5.13)$$

where  $\hat{\mathcal{R}}^s(t)$  is the vector of delivered bitrate to the users of the  $s$ -th SES. To address constraint violations in **P1**, we define penalty function  $p_s^{\text{trans}}$  for transcoding resource constraints:

$$p_s^{\text{trans}}(\hat{\mathcal{R}}^s(t)) = \max \left( 0, \sum_{i=1}^{N_s} \sum_{o=1}^O \eta_o^s \mathbb{R}_i^o - \Omega^s \right). \quad (5.14)$$

We use a centralized critic-actors architecture. After selecting an action and receiving feedback (reward and next state), the critic updates the Temporal Difference (TD) error:

$$\delta^{\pi, \theta} = r_t + \gamma V(\mathbf{x}_{t+1}) - V(\mathbf{x}_t), \quad (5.15)$$

The critic is updated by minimizing the TD difference:

$$V^* = \arg \min_V (\delta^{\pi, \theta})^2, \quad (5.16)$$

Actors are updated using policy gradients:

$$\nabla_{\theta} J(\theta) = E_{\pi_{\theta}} \left[ \nabla_{\theta} \log \pi_{\theta}(o, a) \delta^{\pi, \theta} \right], \quad (5.17)$$

$$\theta \leftarrow \theta + \alpha \nabla_{\theta} \log \pi_{\theta}(o, a) \delta^{\pi, \theta}, \quad (5.18)$$

Figure 5.1 illustrates our proposed architecture. The agent finds proto-actors, expands actions via KNN, and selects the highest Q-value actions for execution. The critic network evaluates  $K$  possible combinations of actions and updates the networks.

The training stages (Algorithm 5.1) initializes the state of the agent and ends at  $T_{\max}$ . Experience tuples  $(s_t, a_t, r_t, s_{t+1})$  are stored in buffer  $\mathcal{M}$ . After training for

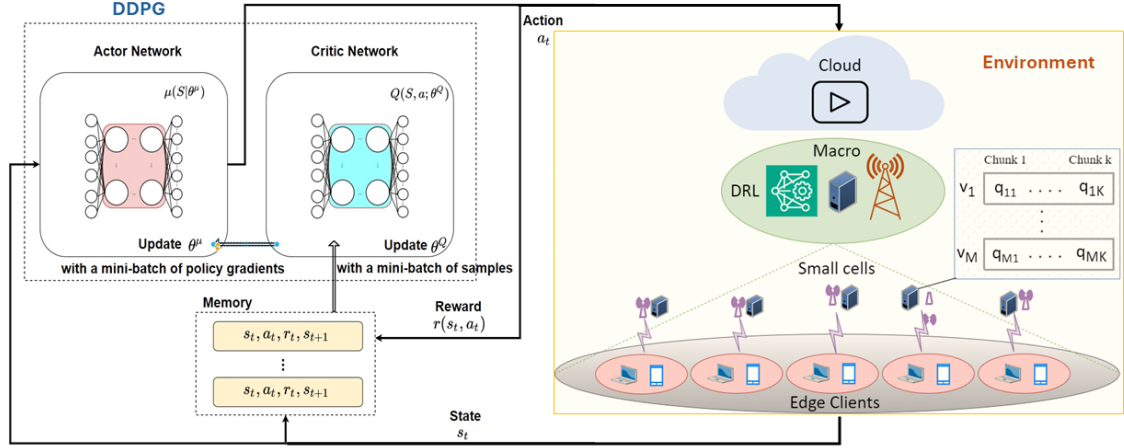


FIGURE 5.1: Our considered architecture

$E_{max}$  episodes, the agent learns the BrA policy.

In testing, the agent loads trained parameters, interacts with the environment, and selects actions based on the output of the actor network.

---

**Algorithm 5.1** Training Stages of the DRL-based Solution

---

**Input:**  $R(t)$  and  $\Phi^{\text{SES}}(t)$ , and  $\forall i \in N$ 

- 1: Initialize the actor and critic networks' parameters randomly.
  - 2: Initialize target networks  $\theta^{\mu'} \leftarrow \theta^\mu$  and  $\theta^{\mathcal{Q}'} \leftarrow \theta^{\mathcal{Q}}$ .
  - 3: Initialize an empty experience memory  $\mathcal{M}$ .
  - 4: **for** each episode  $e = 1, 2, \dots, E_{\max}$  **do**
  - 5:     Generate an initial state  $s_1$  randomly.
  - 6:     **for** each step  $t = 1, 2, \dots, T_{\max}$  **do**
  - 7:         Determine the BrA action  $a_t$  given the demand of the EC, using the current policy network  $\theta^\mu$  and the exploration noise  $\epsilon_\mu$ .
  - 8:         Execute action  $a_t$ , receive the reward  $r_t$  and observe the next state  $s_{t+1}$ .
  - 9:         Store the tuple  $(s_t, a_t, r_t, s_{t+1})$  in replay memory  $\mathcal{M}$ .
  - 10:        Sample a mini-batch of  $K$  tuples from  $\mathcal{M}$ .
  - 11:        Update the critic network by minimizing the loss  $L$  with the samples:
  - 12:        
$$L = \frac{1}{K} \sum_{i=1}^K (r_i + \max_{a \in \mathcal{A}} Q(s'_i, a | \theta^{\mathcal{Q}'}) - Q(s_i, a_i | \theta^{\mathcal{Q}}))^2$$
  - 13:        Update the actor network using the sampled policy gradient:
  - 14:        
$$\nabla_{\theta^\mu} J = \frac{1}{K} \sum_{i=1}^K \nabla_a Q(s_i, a | \theta^{\mathcal{Q}})|_{a=a_i} \nabla_{\theta^\mu} \mu(s_i | \theta^\mu)$$
  - 15:     **end for**
  - 16:     **if**  $t \bmod \delta = 0$  **then**
  - 17:         Update the target networks by  $\theta^{\mu'} \leftarrow \tau \theta^\mu + (1 - \tau) \theta^{\mu'}$  and  $\theta^{\mathcal{Q}'} \leftarrow \tau \theta^{\mathcal{Q}} + (1 - \tau) \theta^{\mathcal{Q}'}$ .
  - 18:     **end if**
  - 19: **end for**
- 

## 5.3 Numerical Results

In this section, we evaluate the performance of the DRL-based algorithm through computer simulations. The simulation setup and results are described in the following subsections.

### 5.3.1 Simulation Setup

A single MBS is located in the center of a circular area with a radius of 500 m, and 5 SBSs are located within the area, each with a radius of 200 m. ECs are distributed in an area with a Poisson distribution with an expected density of  $S/N$  ECs per cell. The buffer length of each EC is considered 90 s [178]. The cache capacity at MES is enough to cache only the highest resolution for all videos (each video will have approximately 4.8 Gb).

We consider 10 videos each having an equal length of 10 minutes [183]. We also consider that different videos have different chunk sizes in the range [4 6] seconds. We select the six most popular resolutions for traditional systems, i.e., 240P, 360P, 480P, 720P, 1080P, and 1440P, each with a bitrate range which is obtained from YouTube similar to [171].

We characterize the popularity of videos using a Zipf-like distribution and sort the videos in  $\mathcal{V}$  in descending order of their popularity  $P_m = \{p_1, \dots, p_i, \dots, p_M\}$ ,  $\sum_{i=1}^M p_i = 1$ , where  $p_m$  represents the popularity of the  $i$ th rank video. Each EC selects a video based on its popularity. In a heterogeneous environment, devices have different interests in terms of resolution selection. Hence, we consider each EC uniformly at random selects one resolution level for its videos, however with a %5 probability for each time slot, the selected resolution downgrades to a lower resolution due to various reasons (e.g., interference, etc.). We also assume ECs with a probability of %10 start streaming from the first chunk, and with a %90 probability select a chunk randomly from the middle of the video in order to show real-world-like user behavior. ECs download the chunks until the play-out buffer is full; such an aggressive approach is also exploited on YouTube [184].

After the USA and the allocation of the bitrates, bandwidth is fairly allocated to the ECs. The chunks are added to the buffer after they are downloaded and the buffer decreases as the chunks are being displayed. For processing capacity, we set  $\Omega^s = 25\text{GHz}$ , which is the maximum number of CPU cycles per second. We also set the number of CPU cycles per Byte 5900 [183].

In the DRL simulation, we use the DDPG agent consisting of two networks: (1) the Actor network has three layers with dimensions 400, 300, and the action dimension, and (2) the Critic network takes as input both the state and action, with layers 400, 300. Both networks use ReLU activations, with Tanh activation in the output of the actor network.

The agent updates its parameters using Adam optimizers, with learning rates of  $1 \times 10^{-4}$  for the actor and  $1 \times 10^{-3}$  for the critic. It uses a replay buffer of size  $1 \times 10^6$  to store experiences and sample mini-batches of size 64 for training. The discount factor ( $\gamma$ ) is set to 0.99, and the soft update coefficient ( $\tau$ ) for the target networks is 0.005. The agent's action selection is deterministic and is bounded by the maximum action value.

### 5.3.2 Simulation Results

To assess the convergence of the DRL-based approach, we conducted an experiment spanning 100 episodes. As illustrated in Figure 5.2, the rewards for both BrA and transcoding converge after approximately 25 episodes of exploration. Similarly, the joint reward (as defined in (5.13)) reaches its maximum value around the same point. Please note that as defined in (5.2), BrA is positive but has been negated since it represents a cost.

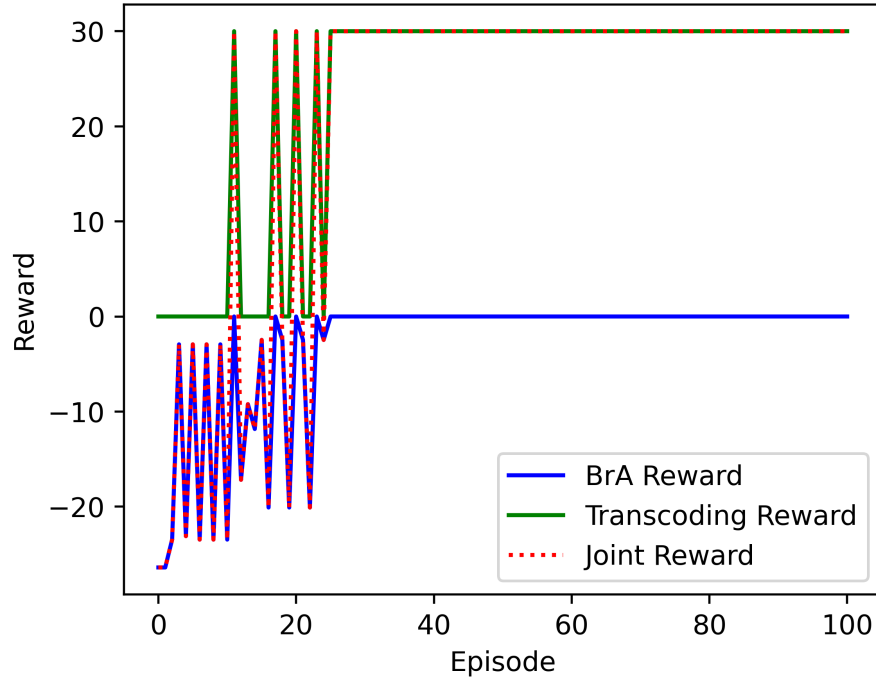


FIGURE 5.2: Bitrate allocation, transcoding and joint rewards vs number of episodes

Figure 5.3.a shows the bitrate error over several episodes, representing the difference between the requested and allocated bitrates. As observed, the error is initially high but drops to zero after approximately 25 episodes, demonstrating the learning and improvement in bitrate allocation over time, which is the primary objective of this study, as defined in **P1**. This trend is further illustrated in Figure 5.3.b, which displays the Root Mean Square of Bitrate Error (RMSBRE), highlighting the reduction in bitrate delivery errors for video chunks over time. In both figures, the proposed solution is compared with a randomized bitrate delivery for clearer contrast.

In order to better understand the source of delivery/streaming and the amount of edge transcoding, we have conducted two experiments, and the results are shown in Figs.5.4 and 5.5. As seen, almost all of the ECs count on streaming from the edge in the first episodes. However, since this violates the transcoding constraint **C1**, the

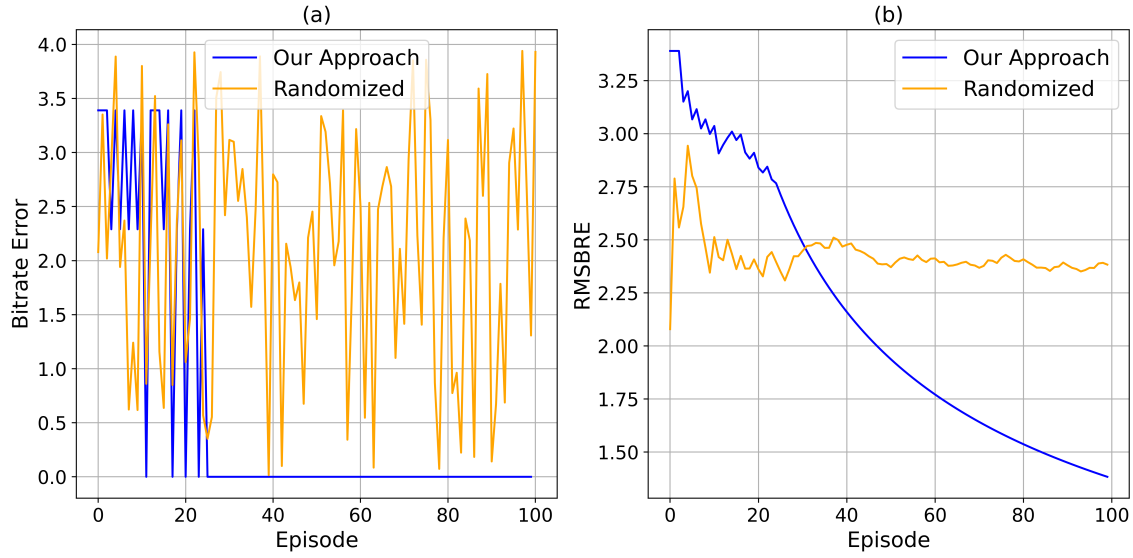


FIGURE 5.3: (a) Bitrate error and (b) Root Mean Square of Bitrate error vs episodes

agent learns this through time and reduces the edge streaming; as a result, edge transcoding, and instead streams more from the MBS and Cloud.

To conclude, the agent initially struggles to deliver the correct bitrate (as shown in Figures 5.2 and 5.3) and exceeds its transcoding capacity, resulting in lower rewards (having a low reward in Figure 5.2). Moreover, as depicted in Figures 5.4 and 5.5 the agent transcodes for most ECs. However, this does not lead to a lower bitrate because the agent transcodes at an incorrect level. Over time, the agent learns to adjust its transcoding process (both when and how much to transcode), reducing the edge transcoding and flexibly utilizing all tiers for efficient streaming.

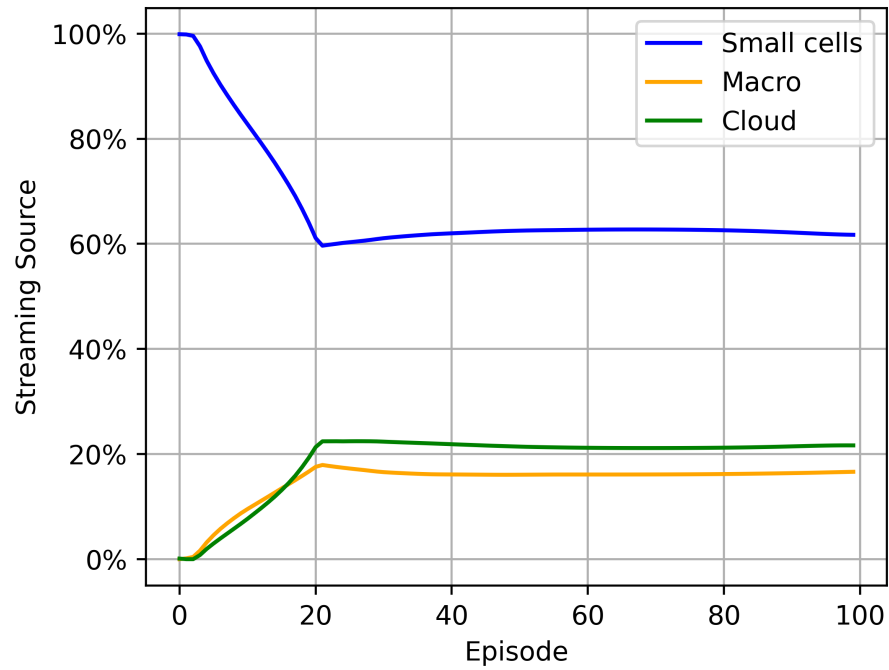


FIGURE 5.5: Streaming source vs number of episodes

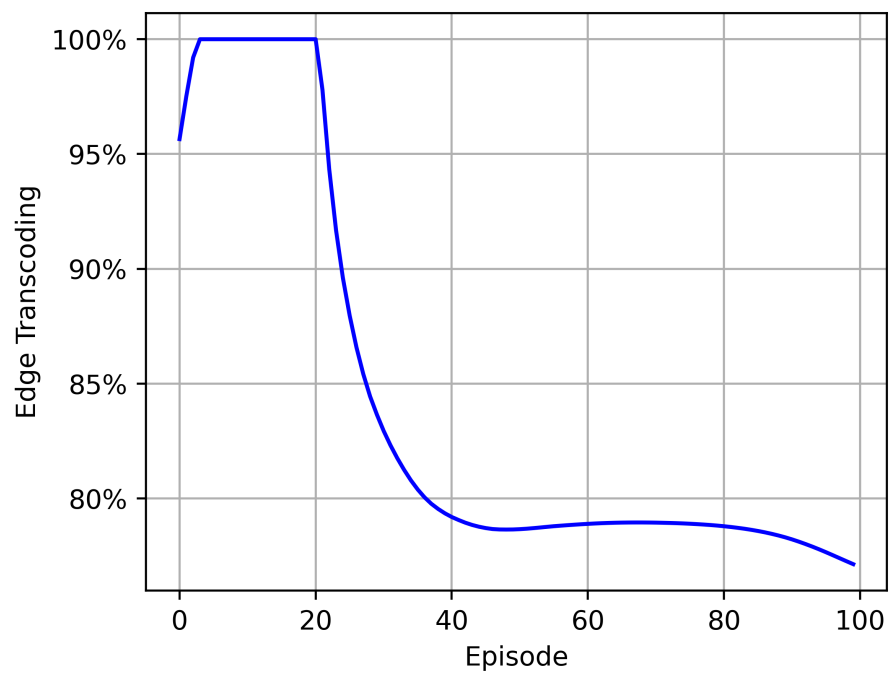


FIGURE 5.4: Transcoding percentage over the edge vs number of episodes



## 5.4 Conclusion

In this work, we proposed a DRL-based solution for the joint optimization of BrA and USA in an Edge-DASH environment. Our DDPG approach significantly improves the system's ability to effectively allocate bitrates while maintaining high QoE standards. The simulation results validate the approach, showing that the agent improves over time, reducing bitrate errors and transcoding violations by selecting appropriate streaming sources, thus balancing the use of edge, macro, and cloud resources. The proposed method demonstrates superior performance compared to traditional solutions, offering a promising strategy for managing video streaming in 5G-enabled multi-tier networks.

# Chapter 6

## Conclusion

Over the course of this dissertation, we have designed, analyzed, and validated a range of distributed machine learning frameworks tailored to the emerging demands of 6G-enabled intelligent Vehicular Networks (VNs). By seamlessly integrating federated learning, split and transfer learning, NS, T/NTNs, and both single-agent and multi-agent DRL, our work demonstrates how privacy-preserving intelligence can be orchestrated across heterogeneous edge, cloud, and non-terrestrial infrastructures. These contributions collectively form a coherent vision for scalable, low-latency, and energy-efficient AI services that support critical vehicular applications—from object detection and adaptive video streaming to coordinated mobility management.

At the heart of our approach lies the paradigm of Distributed Learning-as-a-Service (DLaaS), which virtualizes and orchestrates DL tasks over network slices customized for diverse IoV demands. Treating each slice as a container for specific learning functions, such as federated aggregation, split model training, or Knowledge Distillation (KD), DLaaS dynamically allocates compute and network resources according to real-time requirements. This decoupling of service logic from physical infrastructure

not only ensures that privacy constraints and latency targets are met automatically, but also unlocks new levels of scalability and operational agility.

Building on DLaaS, we developed FSTL and its multilayer generalization (GFSTL), which partition deep models between vehicles and edge or aerial aggregators to minimize communication overhead and accelerate convergence. By leveraging pretrained backbones and adaptively adjusting split points, these schemes achieve robust accuracy under varying mobility and connectivity conditions, effectively bridging the gap between resource-intensive deep networks and the constrained environment of vehicular platforms.

To demonstrate real-world viability, we implemented various DL platforms on resource-constrained hardware—including Raspberry Pi and Odroid devices—and investigated the impact of heterogeneous computation, data distributions, and thermal management on system performance. Warm-start strategies and adaptive client selection in FTL consistently yielded faster convergence, improved model quality, and reduced energy consumption compared to vanilla FL, underscoring the feasibility of in-situ learning across fleets of smart vehicles.

Recognizing the importance of infotainment services, we introduced a multi-agent DRL into an Edge-DASH architecture for adaptive video streaming. Our multi-agent DRL framework enables coordinated bitrate, cache, and bandwidth policies across edge servers, sharing distilled behaviors to reduce exploration overhead and improve resilience to traffic fluctuations. In simulation scenarios that mirror multi-tier edge networks, this approach delivered significant gains in playback smoothness, cache hit rates, and overall user satisfaction compared to static and rule-based baselines.

---

In conclusion, this dissertation paints a cohesive picture of how DL mechanisms—empowered by NS, 6G connectivity, and T/NTNs—can transform vehicular ecosystems. By prioritizing privacy, adaptability, and resource efficiency, our frameworks lay the groundwork for a future in which vehicles not only consume AI services, but also actively shape their evolution.

## *Scope for Future Work*

Looking ahead, several directions stand out for extending this research. One avenue involves enhancing cross-slice intelligence: although DLaaS currently isolates service slices, synergistic learning between applications—such as autonomous navigation, traffic forecasting, and infotainment—could accelerate model convergence and improve generalization. Realizing this vision will require mechanisms for secure intermediate representation sharing, federated multiservice orchestration policies, and privacy-preserving KD across slices.

Another promising direction is the live integration of Non-Terrestrial Networks (NTNs) into federated and split learning workflows. While simulations highlighted the potential of high-altitude platforms and satellites to augment ground-based aggregators, deploying GFSTL over experimental NTN constellations would reveal practical challenges in latency variability, handover management, and orbital dynamics. Such testbeds will be vital for developing synchronization, fault-tolerance, and energy-efficient scheduling strategies under real operational constraints.

Advancing the orchestration intelligence of DLaaS through predictive mobility modeling also offers rich research opportunities. By incorporating fine-grained vehicle trajectory forecasts—derived from GPS traces or onboard sensors—into orchestration engines, the system could proactively reconfigure slices and reassign learning tasks before connectivity degrades. This anticipatory approach demands tight integration between mobility analytics and orchestration modules, supported by efficient online learning algorithms to adapt on the fly.

Ensuring robustness and security in large-scale vehicular federated systems remains paramount. Future work should explore resilient aggregation protocols and

anomaly detection techniques for intermediate representations, alongside practical privacy-amplification methods such as secure multiparty computation or homomorphic encryption. Balancing protection against performance will be critical for preserving trust and integrity in real deployments.

Finally, bridging research to practice will involve engaging with industry consortia and regulators to establish benchmarks, interoperability standards, and data governance frameworks. Comprehensive field trials under diverse regulatory and economic environments will shed light on deployment challenges, paving the way for scalable, ethical, and sustainable AI services in next-generation vehicular networks.



# References

- [1] Carlos Renato Storck and Fátima Duarte-Figueiredo. A survey of 5g technology evolution, standards, and infrastructure associated with vehicle-to-everything communications by internet of vehicles. *IEEE Access*, 8:117593–117614, 2020. doi: 10.1109/ACCESS.2020.3004779.
- [2] Sivarama Prasad Tera, Ravikumar Chinthaginjala, Giovanni Pau, and Tae Hoon Kim. Toward 6g: An overview of the next generation of intelligent network connectivity. *IEEE Access*, 13:925–961, 2025. doi: 10.1109/ACCESS.2024.3523327.
- [3] Elias Dritsas and Maria Trigka. Machine learning in intelligent networks: Architectures, techniques, and use cases. *IEEE Access*, 13:102724–102745, 2025. doi: 10.1109/ACCESS.2025.3577968.
- [4] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021. doi: 10.1109/COMST.2021.3075439.
- [5] Wafa Hamdi, Chahrazed Ksouri, Hasan Bulut, and Mohamed Mosbah. Network slicing-based learning techniques for iov in 5g and beyond networks.



- IEEE Communications Surveys & Tutorials*, 26(3):1989–2047, 2024. doi: 10.1109/COMST.2024.3372083.
- [6] Peng He, Hailong Lei, Dapeng Wu, Ruyan Wang, Yaping Cui, Yan Zhu, and Zhaopeng Ying. Nonterrestrial network technologies: Applications and future prospects. *IEEE Internet of Things Journal*, 12(6):6275–6299, 2025. doi: 10.1109/JIOT.2024.3522912.
- [7] Guozhi Yan, Kai Liu, Chunhui Liu, and Jie Zhang. Edge intelligence for internet of vehicles: A survey. *IEEE Transactions on Consumer Electronics*, 70(2):4858–4877, 2024. doi: 10.1109/TCE.2024.3378509.
- [8] Md. Maruf Hossain Shuvo, Syed Kamrul Islam, Jianlin Cheng, and Bashir I. Morshed. Efficient acceleration of deep learning inference on resource-constrained edge devices: A review. *Proceedings of the IEEE*, 111(1):42–91, 2023. doi: 10.1109/JPROC.2022.3226481.
- [9] Quynh Tu Ngo, Zhifeng Tang, Beeshanga Jayawickrama, Ying He, Eryk Dutkiewicz, and Bathiya Senanayake. Timeliness of information in 5g non-terrestrial networks: A survey. *IEEE Internet of Things Journal*, 11(21):34652–34675, 2024. doi: 10.1109/JIOT.2024.3407105.
- [10] Mohammed S. Elbamby, Cristina Perfecto, Chen-Feng Liu, Jihong Park, Sumudu Samarakoon, Xianfu Chen, and Mehdi Bennis. Wireless edge computing with latency and reliability guarantees. *Proceedings of the IEEE*, 107(8):1717–1737, 2019. doi: 10.1109/JPROC.2019.2917084.
- [11] Tanweer Alam. Data privacy and security in autonomous connected vehicles in smart city environment. *Big Data and Cognitive Computing*, 8(9), 2024. ISSN 2504-2289. doi: 10.3390/bdcc8090095. URL <https://www.mdpi.com/2504-2289/8/9/95>.

- [12] Haitham H. Esmat, Beatriz Lorenzo, and Weisong Shi. Toward resilient network slicing for satellite–terrestrial edge computing iot. *IEEE Internet of Things Journal*, 10(16):14621–14645, 2023. doi: 10.1109/JIOT.2023.3277466.
- [13] Dana Haj Hussein and Mohamed Ibnkahla. Towards intelligent intent-based network slicing for iot systems: Enabling technologies, challenges, and vision. *IEEE Transactions on Network and Service Management*, pages 1–1, 2025. doi: 10.1109/TNSM.2025.3570052.
- [14] Amjad Iqbal, Mau-Luen Tham, Yi Jie Wong, Ala’a Al-Habashna, Gabriel Wainer, Yong Xu Zhu, and Tasos Dagiuklas. Empowering non-terrestrial networks with artificial intelligence: A survey. *IEEE Access*, 11:100986–101006, 2023. doi: 10.1109/ACCESS.2023.3314732.
- [15] Asad Ali, Huang Jianjun, and Ayesha Jabbar. Recent advances in federated learning for connected autonomous vehicles: Addressing privacy, performance, and scalability challenges. *IEEE Access*, 13:80637–80665, 2025. doi: 10.1109/ACCESS.2025.3562128.
- [16] Muhammad Asif Khan, Emna Baccour, Zina Chkirbene, Aiman Erbad, Ridha Hamila, Mounir Hamdi, and Moncef Gabbouj. A survey on mobile edge computing for video streaming: Opportunities and challenges. *IEEE Access*, 10:120514–120550, 2022. doi: 10.1109/ACCESS.2022.3220694.
- [17] David Naseh, Swapnil Sadashiv Shinde, and Daniele Tarchi. Network sliced distributed learning-as-a-service for internet of vehicles applications in 6g non-terrestrial network scenarios. *Journal of Sensor and Actuator Networks*, 13(1), 2024. ISSN 2224-2708. doi: 10.3390/jsan13010014. URL <https://www.mdpi.com/2224-2708/13/1/14>.

- [18] Fengxiao Tang, Bomin Mao, Nei Kato, and Guan Gui. Comprehensive survey on machine learning in vehicular network: Technology, applications and challenges. *IEEE Communications Surveys & Tutorials*, 23(3):2027–2057, 2021. doi: 10.1109/COMST.2021.3089688.
- [19] Jason Posner, Lewis Tseng, Moayad Aloqaily, and Yaser Jararweh. Federated learning in vehicular networks: Opportunities and solutions. *IEEE Network*, 35(2):152–159, 2021. doi: 10.1109/MNET.011.2000430.
- [20] Mingzhe Chen, Deniz Gündüz, Kaibin Huang, Walid Saad, Mehdi Bennis, Aneta Vulgarakis Feljan, and H. Vincent Poor. Distributed learning in wireless networks: Recent progress and future challenges. *IEEE Journal on Selected Areas in Communications*, 39(12):3579–3605, 2021. doi: 10.1109/JSAC.2021.3118346.
- [21] Sijing Duan, Dan Wang, Ju Ren, Feng Lyu, Ye Zhang, Huaqing Wu, and Xuemin Shen. Distributed artificial intelligence empowered by end-edge-cloud computing: A survey. *IEEE Communications Surveys & Tutorials*, 25(1): 591–624, 2023. doi: 10.1109/COMST.2022.3218527.
- [22] Ibrahim Afolabi, Tarik Taleb, Konstantinos Samdanis, Adlen Ksentini, and Hannu Flinck. Network slicing and softwarization: A survey on principles, enabling technologies, and solutions. *IEEE Communications Surveys & Tutorials*, 20(3):2429–2453, 2018. doi: 10.1109/COMST.2018.2815638.
- [23] Zhipeng Cheng, Zhibin Gao, Minghui Liwang, Lianfen Huang, Xiaojiang Du, and Mohsen Guizani. Intelligent task offloading and energy allocation in the uav-aided mobile edge-cloud continuum. *IEEE Network*, 35(5):42–49, 2021. doi: 10.1109/MNET.010.2100025.

- [24] Bodong Shang, Yang Yi, and Lingjia Liu. Computing over space-air-ground integrated networks: Challenges and opportunities. *IEEE Network*, 35(4): 302–309, 2021. doi: 10.1109/MNET.011.2000567.
- [25] Zhi Lin, Min Lin, Benoit Champagne, Wei-Ping Zhu, and Naofal Al-Dhahir. Secrecy-energy efficient hybrid beamforming for satellite-terrestrial integrated networks. *IEEE Transactions on Communications*, 69(9):6345–6360, 2021. doi: 10.1109/TCOMM.2021.3088898.
- [26] Zhi Lin, Hehao Niu, Kang An, Yihua Hu, Dong Li, Jiangzhou Wang, and Naofal Al-Dhahir. Pain without gain: Destructive beamforming from a malicious ris perspective in iot networks. *IEEE Internet of Things Journal*, pages 1–1, 2023. doi: 10.1109/JIOT.2023.3316830.
- [27] Zhi Lin, Hehao Niu, Kang An, Yong Wang, Gan Zheng, Symeon Chatzinotas, and Yihua Hu. Refracting ris-aided hybrid satellite-terrestrial relay networks: Joint beamforming design and optimization. *IEEE Transactions on Aerospace and Electronic Systems*, 58(4):3717–3724, 2022. doi: 10.1109/TAES.2022.3155711.
- [28] Yifu Sun, Kang An, Yonggang Zhu, Gan Zheng, Kai-Kit Wong, Symeon Chatzinotas, Derrick Wing Kwan Ng, and Dongfang Guan. Energy-efficient hybrid beamforming for multilayer ris-assisted secure integrated terrestrial-aerial networks. *IEEE Transactions on Communications*, 70(6):4189–4210, 2022. doi: 10.1109/TCOMM.2022.3170632.
- [29] Prabhat Thakur and Ghanshyam Singh. Cooperative spectrum monitoring in homogeneous and heterogeneous cognitive radio networks. In *Spectrum Sharing in Cognitive Radio Networks: Towards Highly Connected Environments*, pages 121–146. Wiley Telecom, 2021. doi: 10.1002/9781119665458.ch6.

- [30] Prabhat Thakur and Ghanshyam Singh. Radio resource management in internet-of-vehicles. In *Spectrum Sharing in Cognitive Radio Networks: Towards Highly Connected Environments*, pages 311–338. Wiley Telecom, 2021. doi: 10.1002/9781119665458.ch13.
- [31] Prabhat Thakur and Ghanshyam Singh. Interference management in cognitive radio networks. In *Spectrum Sharing in Cognitive Radio Networks: Towards Highly Connected Environments*, pages 255–279. Wiley Telecom, 2021. doi: 10.1002/9781119665458.ch11.
- [32] Priyanka Mishra and Ghanshyam Singh. 6g-iot framework for sustainable smart city: Vision and challenges. In *Sustainable Smart Cities: Enabling Technologies, Energy Trends and Potential Applications*, pages 97–117. Springer International Publishing, Cham, 2023. ISBN 978-3-031-33354-5. doi: 10.1007/978-3-031-33354-5\_5. URL [https://doi.org/10.1007/978-3-031-33354-5\\_5](https://doi.org/10.1007/978-3-031-33354-5_5).
- [33] Shuiguang Deng, Hailiang Zhao, Weijia Fang, Jianwei Yin, Schahram Dustdar, and Albert Y. Zomaya. Edge intelligence: The confluence of edge computing and artificial intelligence. *IEEE Internet of Things Journal*, 7(8):7457–7469, 2020. doi: 10.1109/JIOT.2020.2984887.
- [34] Dimitrios Michael Manias, Ali Chouman, Anwer Al-Dulaimi, and Abdallah Shami. Slice-level performance metric forecasting in intelligent transportation systems and the internet of vehicles. *IEEE Internet of Things Magazine*, 6(3): 56–61, 2023. doi: 10.1109/IOTM.001.2300035.
- [35] Yulei Wu, Hong-Ning Dai, Haozhe Wang, Zehui Xiong, and Song Guo. A survey of intelligent network slicing management for industrial iot: Integrated approaches for smart transportation, smart energy, and smart factory. *IEEE*

- Communications Surveys & Tutorials*, 24(2):1175–1211, 2022. doi: 10.1109/COMST.2022.3158270.
- [36] Mohammed Laroui, Boubakr Nour, Hassine MOUNGLA, Moussa A. Cherif, Hosam Afifi, and Mohsen Guizani. Edge and fog computing for iot: A survey on current research activities & future directions. *Computer Communications*, 180:210–231, 2021. ISSN 0140-3664. doi: 10.1016/j.comcom.2021.09.003.
- [37] Mohammed S. Elbamby, Cristina Perfecto, Chen-Feng Liu, Jihong Park, Sumudu Samarakoon, Xianfu Chen, and Mehdi Bennis. Wireless edge computing with latency and reliability guarantees. *Proceedings of the IEEE*, 107(8):1717–1737, 2019. doi: 10.1109/JPROC.2019.2917084.
- [38] Yiqin Deng, Xianhao Chen, Guangyu Zhu, Yuguang Fang, Zhigang Chen, and Xiaoheng Deng. Actions at the edge: Jointly optimizing the resources in multi-access edge computing. *IEEE Wireless Communications*, 29(2):192–198, 2022. doi: 10.1109/MWC.006.2100699.
- [39] Deze Zeng, Nirwan Ansari, Marie-José Montpetit, Eve M. Schooler, and Daniele Tarchi. Guest editorial: In-network computing: Emerging trends for the edge-cloud continuum. *IEEE Network*, 35(5):12–13, 2021. doi: 10.1109/MNET.2021.9606835.
- [40] Jie Mei, Xianbin Wang, and Kan Zheng. Intelligent network slicing for v2x services toward 5g. *IEEE Network*, 33(6):196–204, 2019. doi: 10.1109/MNET.001.1800528.
- [41] Eugenio Muscinelli, Swapnil Sadashiv Shinde, and Daniele Tarchi. Overview of distributed machine learning techniques for 6G networks. *Algorithms*, 15(6), 2022. ISSN 1999-4893. doi: 10.3390/a15060210. URL <https://www.mdpi.com/1999-4893/15/6/210>.

- 
- [42] Joost Verbraeken, Matthijs Wolting, Jonathan Katzy, Jeroen Kloppenburg, Tim Verbelen, and Jan S. Rellermeyer. A survey on distributed machine learning. *ACM Comput. Surv.*, 53(2):1–33, March 2020. ISSN 0360-0300. doi: 10.1145/3377454. Art. no. 30.
- [43] Yunlong Lu, Xiaohong Huang, Ke Zhang, Sabita Maharjan, and Yan Zhang. Blockchain empowered asynchronous federated learning for secure data sharing in internet of vehicles. *IEEE Transactions on Vehicular Technology*, 69(4): 4298–4311, 2020. doi: 10.1109/TVT.2020.2973651.
- [44] Lorenzo Ridolfi, David Naseh, Swapnil Sadashiv Shinde, and Daniele Tarchi. Implementation and evaluation of a federated learning framework on raspberry pi platforms for iot 6g applications. *Future Internet*, 15(11), 2023. ISSN 1999-5903. doi: 10.3390/fi15110358. URL <https://www.mdpi.com/1999-5903/15/11/358>.
- [45] Yangguang Cui, Kun Cao, Junlong Zhou, and Tongquan Wei. Optimizing training efficiency and cost of hierarchical federated learning in heterogeneous mobile-edge cloud computing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(5):1518–1531, 2023. doi: 10.1109/TCAD.2022.3205551.
- [46] Yuan Zhang, Wenlong Zhang, Lingjun Pu, Tao Lin, and Jinyao Yan. To distill or not to distill: Towards fast, accurate and communication efficient federated distillation learning. *IEEE Internet of Things Journal*, 2023. doi: 10.1109/JIOT.2023.3324666.
- [47] David Naseh, Swapnil Sadashiv Shinde, and Daniele Tarchi. Enabling intelligent vehicular networks through distributed learning in the non-terrestrial networks 6g vision, 2023. URL <https://doi.org/10.48550/arXiv.2310.05899>.

- [48] Mingzhe Chen, H. Vincent Poor, Walid Saad, and Shuguang Cui. Wireless communications for collaborative federated learning. *IEEE Communications Magazine*, 58(12):48–54, 2020. doi: 10.1109/MCOM.001.2000397.
- [49] Caixing Shao, Fengxin Cheng, Jingzhong Xiao, and Ke Zhang. Vehicular intelligent collaborative intersection driving decision algorithm in internet of vehicles. *Future Generation Computer Systems*, 145:384–395, 2023. ISSN 0167-739X. doi: 10.1016/j.future.2023.03.038.
- [50] Rui Song, Dai Liu, Dave Zhenyu Chen, Andreas Festag, Carsten Trinitis, Martin Schulz, and Alois Knoll. Federated learning via decentralized dataset distillation in resource-constrained edge environments. In *2023 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2023. doi: 10.1109/IJCNN54540.2023.10191879.
- [51] Anis Elgabli, Jihong Park, Amrit Singh Bedi, Chaouki Ben Issaid, Mehdi Bennis, and Vaneet Aggarwal. Q-gadmm: Quantized group admm for communication efficient decentralized machine learning. *IEEE Transactions on Communications*, 69(1):164–181, 2021. doi: 10.1109/TCOMM.2020.3026398.
- [52] Shuangshuang Han, Yueyun Chen, Liping Du, and Junwei Lv. Admm-based energy-efficient resource allocation method for internet of vehicles. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 3768–3773, 2022. doi: 10.1109/ITSC55140.2022.9922478.
- [53] Afshin Oroojlooy and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, 53(11):13677–13722, 2023. doi: 10.1007/s10489-022-04105-y.
- [54] Kai Jiang, Huan Zhou, Deze Zeng, and Jie Wu. Multi-agent reinforcement learning for cooperative edge caching in internet of vehicles. In *2020 IEEE*



- 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 455–463, 2020. doi: 10.1109/MASS50613.2020.00062.
- [55] Wei Zhou, Dong Chen, Jun Yan, Zhaojian Li, Huilin Yin, and Wanchen Ge. Multi-agent reinforcement learning for cooperative lane changing of connected and autonomous vehicles in mixed traffic. *Autonomous Intelligent Systems*, 2(1), 2022. doi: 10.1007/s43684-022-00023-5.
- [56] Tong Wang, Azhar Hussain, Lejun Zhang, and Chen Zhao. Collaborative edge computing for social internet of vehicles to alleviate traffic congestion. *IEEE Transactions on Computational Social Systems*, 9(1):184–196, 2022. doi: 10.1109/TCSS.2021.3074038.
- [57] Niccolò Girelli Consolaro, Swapnil Sadashiv Shinde, David Naseh, and Daniele Tarchi. Analysis and performance evaluation of transfer learning algorithms for 6g wireless networks. *Electronics*, 12(15), 2023. ISSN 2079-9292. doi: 10.3390/electronics12153327. URL <https://www.mdpi.com/2079-9292/12/15/3327>.
- [58] Meiyu Wang, Yun Lin, Qiao Tian, and Guangzhen Si. Transfer learning promotes 6g wireless communications: Recent advances and future challenges. *IEEE Transactions on Reliability*, 70(2):790–807, 2021. doi: 10.1109/TR.2021.3062045.
- [59] Zhuoqing Chang, Shubo Liu, Xingxing Xiong, Zhaohui Cai, and Guoqing Tu. A survey of recent advances in edge-computing-powered artificial intelligence of things. *IEEE Internet of Things Journal*, 8(18):13849–13875, 2021. doi: 10.1109/JIOT.2021.3088875.
- [60] Liangkai Liu, Yongtao Yao, Ruijun Wang, Baofu Wu, and Weisong Shi. Equinox: A road-side edge computing experimental platform for cavs. In

- 2020 International Conference on Connected and Autonomous Driving (MetroCAD)*, pages 41–42, 2020. doi: 10.1109/MetroCAD48866.2020.00014.
- [61] Yifan Wang, Liangkai Liu, Xingzhou Zhang, and Weisong Shi. HydraOne: An indoor experimental research and education platform for CAVs. In *2nd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 19)*, Renton, WA, July 2019. USENIX Association. URL <https://www.usenix.org/conference/hotedge19/presentation/wang>.
- [62] Tianze Wu, Yifan Wang, Weisong Shi, and Joshua Lu. Hydramini: An fpga-based affordable research and education platform for autonomous driving. In *2020 International Conference on Connected and Autonomous Driving (MetroCAD)*, pages 45–52, 2020. doi: 10.1109/MetroCAD48866.2020.00016.
- [63] Sumit Maheshwari, Wuyang Zhang, Ivan Seskar, Yanyong Zhang, and Dipankar Raychaudhuri. Edgedrive: Supporting advanced driver assistance systems using mobile edge clouds networks. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–6, 2019. doi: 10.1109/INFOCOMW.2019.8845256.
- [64] Alonica Villanueva, Renzo Leru L. Benemerito, Mark Jetro M. Cabug-Os, Royce B. Chua, Cyrille Kristein D.C. Rebeca, and Menchie Miranda. Somnolence detection system utilizing deep neural network. In *2019 International Conference on Information and Communications Technology (ICOIACT)*, pages 602–607, 2019. doi: 10.1109/ICOIACT46704.2019.8938460.
- [65] Daxin Tian, Chuang Zhang, Xuting Duan, and Xixian Wang. An automatic car accident detection method based on cooperative vehicle infrastructure systems. *IEEE Access*, 7:127453–127463, 2019. doi: 10.1109/ACCESS.2019.2939532.

- [66] Yuchuan Fu, Changle Li, Fei Richard Yu, Tom H. Luan, and Yao Zhang. A survey of driving safety with sensing, vehicular communications, and artificial intelligence-based collision avoidance. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6142–6163, 2022. doi: 10.1109/TITS.2021.3083927.
- [67] Ali Tufail, Abdallah Namoun, Adnan Ahmed Abi Sen, Ki-Hyung Kim, Ahmed Alrehaili, and Arshad Ali. Moisture computing-based internet of vehicles (ioV) architecture for smart cities. *Sensors*, 21(11), 2021. ISSN 1424-8220. doi: 10.3390/s21113785. URL <https://www.mdpi.com/1424-8220/21/11/3785>.
- [68] Junhua Wang, Kun Zhu, and Ekram Hossain. Green internet of vehicles (ioV) in the 6g era: Toward sustainable vehicular communications and networking. *IEEE Transactions on Green Communications and Networking*, 6(1):391–423, 2022. doi: 10.1109/TGCN.2021.3127923.
- [69] Jun Zhang and Khaled B. Letaief. Mobile edge intelligence and computing for the internet of vehicles. *Proceedings of the IEEE*, 108(2):246–261, 2020. doi: 10.1109/JPROC.2019.2947490.
- [70] Wajid Rafique, Lianyong Qi, Ibrar Yaqoob, Muhammad Imran, Raihan Ur Rasool, and Wanchun Dou. Complementing IoT services through software defined networking and edge computing: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):1761–1804, 2020. doi: 10.1109/COMST.2020.2997475.
- [71] Wei Duan, Jinyuan Gu, Miaowen Wen, Guoan Zhang, Yancheng Ji, and Shahid Mumtaz. Emerging technologies for 5g-IOV networks: Applications, trends and opportunities. *IEEE Network*, 34(5):283–289, 2020. doi: 10.1109/MNET.001.1900659.

- 
- [72] Salvador V. Balkus, Honggang Wang, Brian D. Cornet, Chinmay Mahabal, Hieu Ngo, and Hua Fang. A survey of collaborative machine learning using 5g vehicular communications. *IEEE Communications Surveys & Tutorials*, 24(2):1280–1303, 2022. doi: 10.1109/COMST.2022.3149714.
- [73] Penglin Dai, Feng Song, Kai Liu, Yueyue Dai, Pan Zhou, and Songtao Guo. Edge intelligence for adaptive multimedia streaming in heterogeneous internet of vehicles. *IEEE Transactions on Mobile Computing*, 22(3):1464–1478, 2023. doi: 10.1109/TMC.2021.3106147.
- [74] Sahrish Khan Tayyaba, Hasan Ali Khattak, Ahmad Almogren, Munam Ali Shah, Ikram Ud Din, Ibrahim Alkhalifa, and Mohsen Guizani. 5g vehicular network resource management for improving radio access through machine learning. *IEEE Access*, 8:6792–6800, 2020. doi: 10.1109/ACCESS.2020.2964697.
- [75] Swapnil Sadashiv Shinde, Arash Bozorgchenani, Daniele Tarchi, and Qiang Ni. On the design of federated learning in latency and energy constrained computation offloading operations in vehicular edge computing systems. *IEEE Transactions on Vehicular Technology*, 71(2):2041–2057, 2022. doi: 10.1109/TVT.2021.3135332.
- [76] David Naseh, Arash Bozorgchenani, Swapnil Sadashiv Shinde, and Daniele Tarchi. Unified distributed machine learning for 6g intelligent transportation systems: A hierarchical approach for terrestrial and non-terrestrial networks. *Network*, 5(3), 2025. ISSN 2673-8732. doi: 10.3390/network5030041. URL <https://www.mdpi.com/2673-8732/5/3/41>.
- [77] David Naseh, Swapnil Sadashiv Shinde, and Daniele Tarchi. Multi-layer distributed learning for intelligent transportation systems in 6g aerial-ground

- integrated networks. In *2024 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 711–716, 2024. doi: 10.1109/EuCNC/6GSummit60053.2024.10597130.
- [78] David Naseh, Swapnil Sadashiv Shinde, Daniele Tarchi, and Tomaso DeCola. Distributed intelligent framework for remote area observation on multilayer non-terrestrial networks. In *2024 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pages 1–6, 2024. doi: 10.1109/MeditCom61057.2024.10621222.
- [79] David Naseh, Swapnil Sadashiv Shinde, and Daniele Tarchi. Distributed learning framework for earth observation on multilayer non-terrestrial networks. In *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, pages 1–2, 2024. doi: 10.1109/ICMLCN59089.2024.10625007.
- [80] David Naseh, Swapnil Sadashiv Shinde, and Daniele Tarchi. Enabling intelligent vehicular networks through distributed learning in the non-terrestrial networks 6g vision. In *European Wireless 2023; 28th European Wireless Conference*, pages 136–141. VDE, 2023. URL <https://ieeexplore.ieee.org/document/10461444>.
- [81] Fengxiao Tang, Yuichi Kawamoto, Nei Kato, and Jiajia Liu. Future intelligent and secure vehicular network toward 6G: Machine-learning approaches. *Proceedings of the IEEE*, 108(2):292–307, 2020. doi: 10.1109/JPROC.2019.2954595.
- [82] Alberto Attilio Brincat, Federico Pacifici, Stefano Martinaglia, and Francesco Mazzola. The internet of things for intelligent transportation systems in real

- smart cities scenarios. In *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pages 128–132, 2019. doi: 10.1109/WF-IoT.2019.8767247.
- [83] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013. doi: 10.1016/j.future.2013.01.010.
- [84] Tasneem S. J. Darwish and Kamalrulnizam Abu Bakar. Fog based intelligent transportation big data analytics in the internet of vehicles environment: Motivations, architecture, challenges, and critical issues. *IEEE Access*, 6:15679–15701, 2018. doi: 10.1109/ACCESS.2018.2815989.
- [85] Moqbel Hamood, Abdullatif Albaseer, Mohamed Abdallah, Ala Al-Fuqaha, and Amr Mohamed. Optimized federated multitask learning in mobile edge networks: A hybrid client selection and model aggregation approach. *IEEE Transactions on Vehicular Technology*, pages 1–17, 2024. doi: 10.1109/TVT.2024.3427349.
- [86] Huiqiang Chen, Tianqing Zhu, Tao Zhang, Wanlei Zhou, and Philip S. Yu. Privacy and fairness in federated learning: On the perspective of tradeoff. *ACM Comput. Surv.*, 56(2):1–37, Sep. 2023. ISSN 0360-0300. doi: 10.1145/3606017. Article no. 39.
- [87] Yansong Gao, Minki Kim, Chandra Thapa, Alsharif Abuadbba, Zhi Zhang, Seyit Camtepe, Hyounghick Kim, and Surya Nepal. Evaluation and optimization of distributed machine learning techniques for internet of things. *IEEE Transactions on Computers*, 71(10):2538–2552, 2022. doi: 10.1109/TC.2021.3135752.

- [88] Xiaolan Liu, Yansha Deng, and Toktam Mahmoodi. Wireless distributed learning: A new hybrid split and federated learning approach. *IEEE Transactions on Wireless Communications*, 22(4):2650–2665, 2023. doi: 10.1109/TWC.2022.3213411.
- [89] Ricardo Vilalta and Mikhail M. Meskhi. Transfer of knowledge across tasks. In *Metalearning: Applications to Automated Machine Learning and Data Mining*, pages 219–236. Springer International Publishing, Cham, 2022. ISBN 978-3-030-67024-5. doi: 10.1007/978-3-030-67024-5\_12.
- [90] Wanbin Qi, Ronghui Zhang, Jiaen Zhou, Hao Zhang, Yanxi Xie, and Xiaojun Jing. A resource-efficient cross-domain sensing method for device-free gesture recognition with federated transfer learning. *Trans. Green Commun. Netw.*, 7(1):393–400, 2023. doi: 10.1109/TGCN.2022.3233825.
- [91] M. Di Leo, Marco Minghini, Andrej Kóna, Alexander Kotsev, Jean Dusart, Stefanie Lumnitz, C. M. Ilie, Candan Eylül Kilsedar, and Angelos Tzotsos. Digital earth observation infrastructures and initiatives: a review framework based on open principles. *The international archives of the photogrammetry, remote sensing and spatial information sciences*, XLVIII-4/W7-2023:33–40, 2023. doi: 10.5194/isprs-archives-xxviii-4-w7-2023-33-2023.
- [92] Sancho Salcedo-Sanz, Pedram Ghamisi, María Piles, Martin Werner, Lucas Cuadra, A Moreno-Martínez, Emma Izquierdo-Verdiguier, Jordi Muñoz-Marí, Amirhosein Mosavi, and Gustau Camps-Valls. Machine learning information fusion in earth observation: A comprehensive review of methods, applications and data sources. *Information Fusion*, 63:256–272, 2020. doi: 10.1016/j.inffus.2020.07.00.

- 
- [93] Erik Blasch, Tien Pham, Chee-Yee Chong, Wolfgang Koch, Henry Leung, Dave Braines, and Tarek Abdelzaher. Machine learning/artificial intelligence for sensor data fusion—opportunities and challenges. *IEEE Aerospace and Electronic Systems Magazine*, 36(7):80–93, 2021. doi: 10.1109/MAES.2020.3049030.
- [94] Ruiqi Liu, Meng Hua, Ke Guan, Xiping Wang, Leyi Zhang, Tianqi Mao, Di Zhang, Qingqing Wu, and Abbas Jamalipour. 6g enabled advanced transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, 25(9):10564–10580, 2024. doi: 10.1109/TITS.2024.3362515.
- [95] Maadoud Djihane, Hamza Abdelkrim, and Becherif Mohamed. Exploring the potential of non terrestrial networks in next generation intelligent transport systems. In *2024 8th International Conference on Image and Signal Processing and their Applications (ISPA)*, pages 1–7, 2024. doi: 10.1109/ISPA59904.2024.10536838.
- [96] Cheng Dai, Tianli Zhu, Sha Xiang, Lipeng Xie, Sahil Garg, and M. Shamim Hossain. Psfl: Personalized split federated learning framework for distributed model training in intelligent transportation systems. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–10, 2025. doi: 10.1109/TITS.2025.3554710.
- [97] Safa Otoum, Nadra Guizani, and Hussein Mouftah. On the feasibility of split learning, transfer learning and federated learning for preserving security in its systems. *IEEE Transactions on Intelligent Transportation Systems*, 24(7):7462–7470, 2023. doi: 10.1109/TITS.2022.3159092.
- [98] Maoqiang Wu, Guoliang Cheng, Dongdong Ye, Jiawen Kang, Rong Yu, Yuan Wu, and Miao Pan. Federated split learning with data and label privacy preservation in vehicular networks. *IEEE Transactions on Vehicular Technology*, 73



- (1):1223–1238, 2024. doi: 10.1109/TVT.2023.3304176.
- [99] Tommy Kosasi, Zein Adian Laban Sihombing, and Amir Mahmud Hussein. 1. yolo-based vehicle detection: Literature review. *Journal of Computer Networks, Architecture and High Performance Computing*, 2024. doi: 10.47709/cnahpc.v6i3.4377.
- [100] Pian Qi, Diletta Chiaro, Antonella Guzzo, Michele Ianni, Giancarlo Fortino, and Francesco Piccialli. Model aggregation techniques in federated learning: A comprehensive survey. *Future Generation Computer Systems*, 150:272–293, 2024. ISSN 0167-739X. doi: 10.1016/j.future.2023.09.008.
- [101] Mohammed H. Alsharif, Raju Kannadasan, Wei Wei, Kottakkaran Sooppy Nisar, and Abdel-Haleem Abdel-Aty. A contemporary survey of recent advances in federated learning: Taxonomies, applications, and challenges. *Internet of Things*, 27:101251, 2024. ISSN 2542-6605. doi: 10.1016/j.iot.2024.101251.
- [102] Yaochu Jin, Hangyu Zhu, Jinjin Xu, and Yang Chen. *Federated Learning: Fundamentals and Advances*. Machine Learning: Foundations, Methodologies, and Applications. Springer Singapore, 2023. ISBN 978-981-19-7083-2. doi: 10.1007/978-981-19-7083-2.
- [103] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 23(3):1622–1658, 2021. doi: 10.1109/COMST.2021.3075439.
- [104] Zheng Lin, Guanqiao Qu, Xianhao Chen, and Kaibin Huang. Split learning in 6g edge networks. *IEEE Wireless Communications*, 31(4):170–176, 2024. doi: 10.1109/MWC.014.2300319.

- [105] Wen Wu, Mushu Li, Kaige Qu, Conghao Zhou, Xuemin Shen, Weihua Zhuang, Xu Li, and Weisen Shi. Split learning over wireless networks: Parallel design and resource management. *IEEE Journal on Selected Areas in Communications*, 41(4):1051–1066, 2023. doi: 10.1109/JSAC.2023.3242704.
- [106] Praneeth Vepakomma and Ramesh Raskar. Split learning: A resource efficient model and data parallel approach for distributed deep learning. In Heiko Ludwig and Nathalie Baracaldo, editors, *Federated Learning: A Comprehensive Overview of Methods and Applications*, pages 439–451. Springer International Publishing, Cham, 2022. ISBN 978-3-030-96896-0. doi: 10.1007/978-3-030-96896-0\_19. URL [https://doi.org/10.1007/978-3-030-96896-0\\_19](https://doi.org/10.1007/978-3-030-96896-0_19).
- [107] Valeria Turina, Zongshun Zhang, Flavio Esposito, and Ibrahim Matta. Federated or split? a performance and privacy analysis of hybrid split and federated learning architectures. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 250–260, 2021. doi: 10.1109/CLOUD53861.2021.00038.
- [108] Yilei Liang, Pranvera Kortoçi, Pengyuan Zhou, Lik-Hang Lee, Abbas Mehrabi, Pan Hui, Sasu Tarkoma, and Jon Crowcroft. Federated split gans for collaborative training with heterogeneous devices. *Software Impacts*, 14:100436, 2022. ISSN 2665-9638. doi: 10.1016/j.simpa.2022.100436.
- [109] David Naseh, Mahdi Abdollahpour, and Daniele Tarchi. Real-world implementation and performance analysis of distributed learning frameworks for 6g iot applications. *Information*, 15(4), 2024. ISSN 2078-2489. doi: 10.3390/info15040190. URL <https://www.mdpi.com/2078-2489/15/4/190>.

- [110] Zehui Zhao, Laith Alzubaidi, Jinglan Zhang, Ye Duan, and Yuantong Gu. A comparison review of transfer learning and self-supervised learning: Definitions, applications, advantages and limitations. *Expert Systems with Applications*, 242:122807, 2024. ISSN 0957-4174. doi: 10.1016/j.eswa.2023.122807.
- [111] Zhisheng Niu, Xuemin S. Shen, Qinyu Zhang, and Yuliang Tang. Space-air-ground integrated vehicular network for connected and automated vehicles: Challenges and solutions. *Intelligent and Converged Networks*, 1(2):142–169, 2020. doi: 10.23919/ICN.2020.0009.
- [112] Glenn Jocher. Ultralytics yolov5 (version 7.0), 2020. URL <https://github.com/ultralytics/yolov5>.
- [113] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016. doi: 10.1109/CVPR.2016.350.
- [114] Rahima Khanam and Muhammad Hussain. What is yolov5: A deep look into the internal features of the popular object detector, 2024. URL <https://arxiv.org/abs/2407.20892>.
- [115] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38:100857, 2023. ISSN 2210-5379. doi: <https://doi.org/10.1016/j.suscom.2023.100857>. URL <https://www.sciencedirect.com/science/article/pii/S2210537923000124>.

- [116] Khaled B. Letaief, Yuanming Shi, Jianmin Lu, and Jianhua Lu. Edge artificial intelligence for 6G: Vision, enabling technologies, and applications. *IEEE Journal on Selected Areas in Communications*, 40(1):5–36, 2022. doi: 10.1109/JSAC.2021.3126076.
- [117] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, Dusit Niyato, Octavia Dobre, and H. Vincent Poor. 6G Internet of Things: A comprehensive survey. *IEEE Internet of Things Journal*, 9(1):359–383, 2022. doi: 10.1109/JIOT.2021.3103320.
- [118] Claudio Paoloni. 6G technology overview - second edition. White paper, one6G, November 2022. <https://one6g.org/download/2699/>.
- [119] Fengxiao Tang, Bomin Mao, Yuichi Kawamoto, and Nei Kato. Survey on machine learning for intelligent end-to-end communication toward 6G: From network access, routing to traffic control and streaming adaption. *IEEE Communications Surveys & Tutorials*, 23(3):1578–1598, 2021. doi: 10.1109/COMST.2021.3073009.
- [120] Eugenio Muscinelli, Swapnil Sadashiv Shinde, and Daniele Tarchi. Overview of distributed machine learning techniques for 6g networks. *Algorithms*, 15(6), 2022. ISSN 1999-4893. doi: 10.3390/a15060210.
- [121] Swapnil Sadashiv Shinde, Arash Bozorgchenani, Daniele Tarchi, and Qiang Ni. On the design of federated learning in latency and energy constrained computation offloading operations in vehicular edge computing systems. *IEEE Transactions on Vehicular Technology*, 71(2):2041–2057, 2022. doi: 10.1109/TVT.2021.3135332.
- [122] Swapnil Sadashiv Shinde and Daniele Tarchi. Joint air-ground distributed federated learning for intelligent transportation systems. *IEEE Transactions*

- on Intelligent Transportation Systems*, 24(9):9996–10011, 2023. doi: 10.1109/TITS.2023.3265416.
- [123] Qiang Duan, Jun Huang, Shijing Hu, Ruijun Deng, Zhihui Lu, and Shui Yu. Combining federated learning and edge computing toward ubiquitous intelligence in 6g network: Challenges, recent advances, and future directions. *IEEE Communications Surveys & Tutorials*, 25(4):2892–2950, 2023. doi: 10.1109/COMST.2023.3316615.
- [124] Manuel Eugenio Morocho-Cayamcela, Haeyoung Lee, and Wansu Lim. Machine learning for 5G/B5G mobile and wireless communications: Potential, limitations, and future directions. *IEEE Access*, 7:137184–137206, 2019. doi: 10.1109/ACCESS.2019.2942390.
- [125] D. Praveen Kumar, Tarachand Amgoth, and Chandra Sekhara Rao Annavarapu. Machine learning algorithms for wireless sensor networks: A survey. *Information Fusion*, 49:1–25, 2019. ISSN 1566-2535. doi: 10.1016/j.inffus.2018.09.013.
- [126] G Fontanesi, F Ortíz, E Lagunas, V Monzon Baeza, MÁ Vázquez, JA Vásquez-Peralvo, M Minardi, HN Vu, PJ Honnaiah, C Lacoste, et al. Artificial intelligence for satellite communication and non-terrestrial networks: A survey, 2023. URL <https://arxiv.org/abs/2304.13008>.
- [127] Hoon Lee, Sang Hyun Lee, and Tony Q. S. Quek. Deep learning for distributed optimization: Applications to wireless resource management. *IEEE Journal on Selected Areas in Communications*, 37(10):2251–2266, 2019. doi: 10.1109/JSAC.2019.2933890.
- [128] Jiwei Huang, Jiangyuan Wan, Bofeng Lv, Qiang Ye, and Ying Chen. Joint computation offloading and resource allocation for edge-cloud collaboration in

- internet of vehicles via deep reinforcement learning. *IEEE Systems Journal*, 17(2):2500–2511, 2023. doi: 10.1109/JSYST.2023.3249217.
- [129] Hao Song, Lingjia Liu, Jonathan Ashdown, and Yang Yi. A deep reinforcement learning framework for spectrum management in dynamic spectrum access. *IEEE Internet of Things Journal*, 8(14):11208–11218, 2021. doi: 10.1109/JIOT.2021.3052691.
- [130] Padmalaya Nayak, G.K. Swetha, Surbhi Gupta, and K. Madhavi. Routing in wireless sensor networks using machine learning techniques: Challenges and opportunities. *Measurement*, 178, 2021. ISSN 0263-2241. doi: 10.1016/j.measurement.2021.108974. URL <https://www.sciencedirect.com/science/article/pii/S0263224121000117>.
- [131] Zhaohui Yang, Mingzhe Chen, Walid Saad, Choong Seon Hong, and Mohammad Shikh-Bahaei. Energy efficient federated learning over wireless communication networks. *IEEE Transactions on Wireless Communications*, 20(3):1935–1949, 2021. doi: 10.1109/TWC.2020.3037554.
- [132] Ji Chu Jiang, Burak Kantarci, Sema Oktug, and Tolga Soyata. Federated learning in smart city sensing: Challenges and opportunities. *Sensors*, 20(21), 2020. ISSN 1424-8220. doi: 10.3390/s20216230. URL <https://www.mdpi.com/1424-8220/20/21/6230>.
- [133] Bho Matthiesen, Nasrin Razmi, Israel Leyva-Mayorga, Armin Dekorsy, and Petar Popovski. Federated learning in satellite constellations. *IEEE Network*, pages 1–16, 2023. doi: 10.1109/MNET.132.2200504. Early Access.

- [134] Muhammad Usman Younus, Muhammad Khurram Khan, and Abdul Rauf Bhatti. Improving the software-defined wireless sensor networks routing performance using reinforcement learning. *IEEE Internet of Things Journal*, 9(5):3495–3508, 2022. doi: 10.1109/JIOT.2021.3102130.
- [135] Deepak Kumar Dewangan and Satya Prakash Sahu. Deep learning-based speed bump detection model for intelligent vehicle system using Raspberry Pi. *IEEE Sensors Journal*, 21(3):3570–3578, 2021. doi: 10.1109/JSEN.2020.3027097.
- [136] Giovanni Cicceri, Giuseppe Tricomi, Zakaria Benomar, Francesco Longo, Antonio Puliafito, and Giovanni Merlino. Dilocc: An approach for across the computing continuum. In *2021 IEEE International Conference on Smart Computing (SMARTCOMP)*, pages 113–120, 2021. doi: 10.1109/SMARTCOMP52413.2021.00036.
- [137] Jed Mills, Jia Hu, and Geyong Min. Communication-efficient federated learning for wireless edge intelligence in IoT. *IEEE Internet of Things Journal*, 7(7):5986–5994, 2020. doi: 10.1109/JIOT.2019.2956615.
- [138] Attila Farkas, Gábor Kertész, and Róbert Lovas. Parallel and distributed training of deep neural networks: A brief overview. In *2020 IEEE 24th International Conference on Intelligent Engineering Systems (INES)*, pages 165–170, 2020. doi: 10.1109/INES49302.2020.9147123.
- [139] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020. doi: 10.1109/MSP.2020.2975749.
- [140] Choong Seon Hong, Latif U Khan, Mingzhe Chen, Dawei Chen, Walid Saad, and Zhu Han. *Federated learning for wireless networks*. Springer, Singapore, 2022.

- [141] Zhao Zhang, Yong Zhang, Da Guo, Shuang Zhao, and Xiaolin Zhu. Communication-efficient federated continual learning for distributed learning system with non-iid data. *Science China Information Sciences*, 66(2), 2023. doi: 10.1007/s11432-020-3419-4.
- [142] Daniel J. Beutel, Taner Topal, Akhil Mathur, Xinchu Qiu, Javier Fernandez-Marques, Yan Gao, Lorenzo Sani, Kwing Hei Li, Titouan Parcollet, Pedro Porto Buarque de Gusmão, and Nicholas D. Lane. Flower: A friendly federated learning research framework, 2022. URL <https://arxiv.org/abs/2007.14390>.
- [143] Ensheng Liu, Liping Zheng, Qiang He, Phu Lai, Benzhu Xu, and Gaofeng Zhang. Role-based user allocation driven by criticality in edge computing. *IEEE Transactions on Services Computing*, 16(5):3636–3650, 2023. doi: 10.1109/TSC.2023.3280498.
- [144] Qiong Wu, Xiaobo Wang, Qiang Fan, Pingyi Fan, Cui Zhang, and Zhengquan Li. High stable and accurate vehicle selection scheme based on federated edge learning in vehicular networks. *China Communications*, 20(3):1–17, 2023. doi: 10.23919/JCC.2023.03.001.
- [145] Latif U. Khan, Ehaz Mustafa, Junaid Shuja, Faisal Rehman, Kashif Bilal, Zhu Han, and Choong Seon Hong. Federated learning for digital twin-based vehicular networks: Architecture and challenges. *IEEE Wireless Communications*, pages 1–8, 2023. doi: 10.1109/MWC.012.2200373.
- [146] Zemin Sun, Geng Sun, Yanheng Liu, Jian Wang, and Dongpu Cao. Bargain-match: A game theoretical approach for resource allocation and task offloading in vehicular edge computing networks. *IEEE Transactions on Mobile Computing*, 23(2):1655–1673, 2024. doi: 10.1109/TMC.2023.3239339.



- [147] Shuai Wang, Yuncong Hong, Rui Wang, Qi Hao, Yik-Chung Wu, and Derrick Wing Kwan Ng. Edge federated learning via unit-modulus over-the-air computation. *IEEE Transactions on Communications*, 70(5):3141–3156, 2022.
- [148] Wei-Bin Kou, Shuai Wang, Guangxu Zhu, Bin Luo, Yingxian Chen, Derrick Wing Kwan Ng, and Yik-Chung Wu. Communication resources constrained hierarchical federated learning for end-to-end autonomous driving. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9383–9390. IEEE, 2023.
- [149] Dingzhu Wen, Peixi Liu, Guangxu Zhu, Yuanming Shi, Jie Xu, Yonina C Eldar, and Shuguang Cui. Task-oriented sensing, computation, and communication integration for multi-device edge ai. *IEEE Transactions on Wireless Communications*, 2023.
- [150] Ziyao Liu, Jiale Guo, Wenzhuo Yang, Jiani Fan, Kwok-Yan Lam, and Jun Zhao. Privacy-preserving aggregation in federated learning: A survey. *IEEE Transactions on Big Data*, pages 1–20, 2022. doi: 10.1109/TBDATA.2022.3190835.
- [151] Taki Hasan Rafi, Faiza Anan Noor, Tahmid Hussain, and Dong-Kyu Chae. Fairness and privacy preserving in federated learning: A survey. *Information Fusion*, 105:102198, 2024. ISSN 1566-2535. doi: <https://doi.org/10.1016/j.inffus.2023.102198>. URL <https://www.sciencedirect.com/science/article/pii/S1566253523005146>.
- [152] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *arXiv preprint arXiv:1909.06335*, 2019.

- [153] Huachi Zhou, Junhong Cheng, Xiangfeng Wang, and Bo Jin. Low rank communication for federated learning. In *Database Systems for Advanced Applications. DASFAA 2020 International Workshops: BDMS, SeCoP, BDQM, GDMA, and AIDE, Jeju, South Korea, September 24–27, 2020, Proceedings 25*, pages 1–16. Springer, 2020.
- [154] Jianyu Wang, Qinghua Liu, Hao Liang, Gauri Joshi, and H Vincent Poor. A novel framework for the analysis and design of heterogeneous federated learning. *IEEE Transactions on Signal Processing*, 69:5234–5249, 2021.
- [155] Xin Yao and Lifeng Sun. Continual local training for better initialization of federated models. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1736–1740. IEEE, 2020.
- [156] Qinbin Li, Bingsheng He, and Dawn Song. Model-contrastive federated learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10713–10722, 2021.
- [157] Canh T. Dinh, Nguyen H. Tran, Minh N. H. Nguyen, Choong Seon Hong, Wei Bao, Albert Y. Zomaya, and Vincent Gramoli. Federated learning over wireless networks: Convergence analysis and resource allocation. *IEEE/ACM Transactions on Networking*, 29(1):398–409, 2021. doi: 10.1109/TNET.2020.3035770.
- [158] Zonghang Li, Yihong He, Hongfang Yu, Jiawen Kang, Xiaoping Li, Zenglin Xu, and Dusit Niyato. Data heterogeneity-robust federated learning via group client selection in industrial iot. *IEEE Internet of Things Journal*, 9(18): 17844–17857, 2022. doi: 10.1109/JIOT.2022.3161943.
- [159] Kai Jiang, Yue Cao, Yujie Song, Huan Zhou, Shaohua Wan, and Xu Zhang. Asynchronous federated and reinforcement learning for mobility-aware edge

- caching in iovs. *IEEE Internet of Things Journal*, pages 1–1, 2024. doi: 10.1109/JIOT.2023.3349255.
- [160] S. Amiri, A. Belloum, E. Nalisnick, S. Klous, and L. Gommans. On the impact of non-iid data on the performance and fairness of differentially private federated learning. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 52–58, Los Alamitos, CA, USA, jun 2022. IEEE Computer Society. doi: 10.1109/DSN-W54100.2022.00018. URL <https://doi.ieeecomputersociety.org/10.1109/DSN-W54100.2022.00018>.
- [161] Yang Liu, Yan Kang, Chaoping Xing, Tianjian Chen, and Qiang Yang. A secure federated transfer learning framework. *IEEE Intelligent Systems*, 35(4):70–82, 2020. doi: 10.1109/MIS.2020.2988525.
- [162] Hongwei Yang, Hui He, Weizhe Zhang, and Xiaochun Cao. Fedsteg: A federated transfer learning framework for secure image steganalysis. *IEEE Transactions on Network Science and Engineering*, 8(2):1084–1094, 2021. doi: 10.1109/TNSE.2020.2996612.
- [163] Shreya Sharma, Chaoping Xing, Yang Liu, and Yan Kang. Secure and efficient federated transfer learning. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 2569–2576, 2019. doi: 10.1109/BigData47090.2019.9006280.
- [164] Aili Wang, Yutong Zhang, and Yixin Yan. Heterogeneous defect prediction based on federated transfer learning via knowledge distillation. *IEEE Access*, 9:29530–29540, 2021. doi: 10.1109/ACCESS.2021.3058886.
- [165] Dashan Gao, Yang Liu, Anbu Huang, Ce Ju, Han Yu, and Qiang Yang. Privacy-preserving heterogeneous federated transfer learning. In *2019 IEEE*

- International Conference on Big Data (Big Data)*, pages 2552–2559, 2019. doi: 10.1109/BigData47090.2019.9005992.
- [166] Zhigang Yan and Dong Li. Performance analysis for resource constrained decentralized federated learning over wireless networks. *IEEE Transactions on Communications*, pages 1–1, 2024. doi: 10.1109/TCOMM.2024.3362143.
- [167] Muhammad Shiraz, Saeid Abolfazli, Zohreh Sanaei, and Abdullah Gani. A study on virtual machine deployment for application outsourcing in mobile cloud computing. *The Journal of Supercomputing*, 63:946–964, 2013. doi: 10.1007/s11227-012-0846-y.
- [168] David Naseh, Arash Bozorgchenani, and Daniele Tarchi. Deep reinforcement learning for edge-dash-based dynamic video streaming. In *2025 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2025. doi: 10.1109/WCNC61545.2025.10978132.
- [169] *Ericsson Mobility Reports*. Ericsson, Nov 2021.
- [170] Shashwat Kumar, Doddala Sai Vineeth, and Antony Franklin A. Edge assisted DASH video caching mechanism for multi-access edge computing. In *2018 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6, Indore, India, 2018. doi: 10.1109/ANTS.2018.8710106.
- [171] D. Wang, Y. Peng, X. Ma, W. Ding, H. Jiang, F. Chen, and J. Liu. Adaptive wireless video streaming based on edge computing: Opportunities and approaches. *IEEE Transactions on Services Computing*, 12(5):685–697, 2019. doi: 10.1109/TSC.2018.2828426.

- 
- [172] Majd Nafeh, Arash Bozorgchenani, and Daniele Tarchi. Joint scalable video coding and transcoding solutions for fog-computing-assisted DASH video applications. *Future Internet*, 14(9), September 2022. ISSN 1999-5903. doi: 10.3390/fi14090268.
- [173] A. Bozorgchenani, D. Tarchi, and G. E. Corazza. Centralized and distributed architectures for energy and delay efficient fog network-based edge computing services. *IEEE Trans. Green Commun. Netw.*, 3(1):250–263, 2019. doi: 10.1109/TGCN.2018.2885443.
- [174] Arash Bozorgchenani, Farshad Mashhadi, Daniele Tarchi, and Sergio A. Salinas Monroy. Multi-objective computation sharing in energy and delay constrained mobile edge computing environments. *IEEE Transactions on Mobile Computing*, 20(10):2992–3005, 2021. doi: 10.1109/TMC.2020.2994232.
- [175] H. A. Pedersen and S. Dey. Enhancing mobile video capacity and quality using rate adaptation, RAN caching and processing. *IEEE/ACM Transactions on Networking*, 24(2):996–1010, 2016. doi: 10.1109/TNET.2015.2410298.
- [176] A. Mehrabi, M. Siekkinen, and A. Ylä-Jääski. Edge computing assisted adaptive mobile video streaming. *IEEE Transactions on Mobile Computing*, 18(4):787–800, 2019. doi: 10.1109/TMC.2018.2850026.
- [177] S. Bayhan, S. Maghsudi, and A. Zubow. EdgeDASH: Exploiting network-assisted adaptive video streaming for edge caching. *IEEE Transactions on Network and Service Management*, 18(2):1732–1745, 2021. doi: 10.1109/TNSM.2020.3037147.
- [178] L. Tao, Y. Gong, S. Jin, and J. Zhao. Energy-efficient predictive HTTP adaptive streaming in mobile cellular networks. *IEEE Transactions on Vehicular Technology*, 67(11):11069–11083, 2018. doi: 10.1109/TVT.2018.2868791.

- [179] Z. Yan, J. Xue, and C. W. Chen. Prius: Hybrid edge cloud and client adaptation for HTTP adaptive streaming in cellular networks. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(1):209–222, 2017. doi: 10.1109/TCSVT.2016.2539827.
- [180] W. Zhang, Y. Wen, Z. Chen, and A. Khisti. QoE-driven cache management for HTTP adaptive bit rate streaming over wireless networks. *IEEE Transactions on Multimedia*, 15(6):1431–1445, 2013. doi: 10.1109/TMM.2013.2247583.
- [181] P. Reichl, B. Tuffin, and R. Schatz. Logarithmic laws in service quality perception: where microeconomics meets psychophysics and quality of experience. *Telecommunication Systems*, 52(2):587–600, 2013. doi: 10.1007/s11235-011-9503-7.
- [182] Gabriel Dulac-Arnold, Richard Evans, Hado van Hasselt, Peter Sunehag, Timothy Lillicrap, Jonathan Hunt, Timothy Mann, Theophane Weber, Thomas Degris, and Ben Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*, 2015.
- [183] S. Rezvani, S. Parsaeefard, N. Mokari, M. R. Javan, and H. Yanikomeroglu. Cooperative multi-bitrate video caching and transcoding in multicarrier NOMA-assisted heterogeneous virtualized MEC networks. *IEEE Access*, 7: 93511–93536, 2019. doi: 10.1109/ACCESS.2019.2927903.
- [184] Hans W. Barz and Gregory A. Bassett. *Multimedia Networks: Protocols, Design and Applications*. Wiley, 2016.