National Technical University of Athens
School of Electrical and Computer Engineering
Division of Information Transmission Systems & Material Technology

Federated Machine Learning in Network Environments for Connected and Automated Mobility Applications

Ph.D. Thesis

GEORGIOS DRAINAKIS

Dipl.-Ing. in Electrical and Computer Engineering, NTUA

Supervisor: Prof. Dimitra-Theodora I. Kaklamani

Athens, December 2024

Εθνικο Μετσοβιο Πολυτεχνειο

Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Συστηματων Μεταδοσης Πληροφοριας & Τεχνολογιας Υλικων

Συνεργατική Μηχανική Μάθηση σε Δικτυακά Περιβάλλοντα και Εφαρμογές σε Συστήματα Αυτόνομης και Διασυνδεδεμένης Κινητικότητας

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

Γεώργιου Δραϊνάκη

Διπλωματούχου Ηλεκτρολόγου Μηχανικού & Μηχανικού Υπολογιστών Ε.Μ.Π.

Επιβλέπουσα: Δήμητρα-Θεοδώρα Ι. Κακλαμάνη, Καθηγήτρια Ε.Μ.Π.

Αθήνα, Δεκέμβριος 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Συστημάτων Μετάδοσης Πληροφορίας & Τεχνολογίας Υλικών

Συνεργατική Μηχανική Μάθηση σε Δικτυακά Περιβάλλοντα και Εφαρμογές σε Συστήματα Αυτόνομης και Διασυνδεδεμένης Κινητικότητας

ΔΙΔΑΚΤΟΡΙΚΗ ΔΙΑΤΡΙΒΗ

του

Γεώργιου Δραϊνάκη

Διπλωματούχου Ηλεκτρολόγου Μηχανικού και Μηχανικού Υπολογιστών Ε.Μ.Π.

Συμβουλευτική Επιτροπή: Δήμητρα-Θεοδώρα Ι. Κακλαμάνη, Καθηγήτρια
Ιάκωβος Στ. Βενιέρης, Καθηγητής
Άγγελος Αμδίτης, Ερευνητής Α΄

Εγκρίθηκε από την επταμελή εξεταστική επιτροπή την 5η Δεκεμβρίου 2024.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Δήμητρα-Θεοδώρα Ι. Κακλαμάνη	Ιάκωβος Στ. Βενιέρης	Άγγελος Αμδίτης
Καθηγήτρια	Καθηγητής	Ερευνητής Α΄
Ε.Μ.Π.	Ε.Μ.Π.	ΕΠΙΣΕΥ-Ε.Μ.Π.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....	.....
Αθανάσιος Παναγόπουλος	Εμμανουήλ Βαρβαρίγος	Γεώργιος Στάμου	Παναγιώτης Γκόνης
Καθηγητής	Καθηγητής	Καθηγητής	Επίκουρος Καθηγητής
Ε.Μ.Π.	Ε.Μ.Π.	Ε.Μ.Π.	ΕΚΠΑ

Αθήνα, Δεκέμβριος 2024

(Υπογραφή)

.....
Γεώργιος Δραϊνάκης
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά την επιβλέπουσά μου κ. Δήμητρα Κακλαμάνη για την υποστήριξη και την επίβλεψη σε όλη τη δύσκολη διαδρομή της εκπόνησης της διατριβής, καθώς και όλα τα παιδιά από τα εργαστήρια Μικροκυμάτων και Οπτικών Ινών και ICBNet του ΕΜΠ και ιδιαίτερα τους Γιάννη Μπαρτσιώκα και Χαρά Ψάρρη, για την πολύτιμη βοήθειά τους.

Ιδιαίτερες ευχαριστίες εκφράζονται τόσο στην τριμελή συμβουλευτική επιτροπή όσο και στην επταμελή εξεταστική επιτροπή για τις χρήσιμες συμβουλές και παρατηρήσεις τους.

Ξεχωριστά θα ήθελα να ευχαριστήσω όλα τα παιδιά από το εργαστήριο του ΕΠΙΣΕΥ που με βοήθησαν κατά τη διάρκεια του διδακτορικού, ξεκινώντας από τους Παναγιώτη Πανταζόπουλο, Κωνσταντίνο Κατσαρό και Βασίλη Σούρλα που παρείχαν την επιστημονική καθοδήγηση από τη αρχή αυτής της προσπάθειας μέχρι και την ολοκλήρωση της ερευνητικής διαδικασίας, και έπειτα τους παλαιότερους ΥΔ και συναδέλφους για την τεχνική και κυρίως την ψυχολογική βοήθεια: τον Μάρκο Αντωνόπουλο, την Κατερίνα Αργύρη, τον Θοδωρή Θεοδωρόπουλο, τον Παύλο Μπασαρά, την Αναστασία Μπολοβίνου, τον Αργύρη Ρουμελιώτη και τον Γιώργο Χατζηπαυλή.

Τέλος, ευχαριστώ θερμά την οικογένειά μου που με στήριξε καθόλη τη διάρκεια και μου έδινε κουράγιο, στους οποίους και αφιερώνεται η διατριβή.

Περίληψη

Τα δίκτυα Πέμπτης και επόμενης Γενιάς (Beyond 5G - B5G) αναμένεται να μεταμορφώσουν τις κινητές επικοινωνίες, επιτρέποντας την ενσωμάτωση ανθρώπων, συσκευών (Internet of Things - IoT) και αισθητήρων σε ψηφιακά-φυσικά περιβάλλοντα, πραγματοποιώντας τελικά την έννοια του Υπερδικτύου των Πραγμάτων (Internet of Everything - IoE). Ένας βασικός παράγοντας που διευκολύνει αυτή τη μετάβαση είναι η σύγκλιση της Υπολογιστικής Νέφους (cloud computing) προς το πεδίο των κινητών συσκευών, που αναφέρεται ως Υπολογιστικό Συνεχές (compute continuum). Αυτό διευκολύνει την ανταλλαγή δεδομένων, την επεξεργασία και τη λήψη αποφάσεων σε όλους τους τομείς του δικτύου, από το νέφος και την άκρη του δικτύου (Edge), φτάνοντας έως και τις συσκευές σε επίπεδο χρήστη (Extreme-Edge).

Η Τεχνητή Νοημοσύνη και η Μηχανική Μάθηση (Artificial Intelligence/Machine Learning - AI/ML) παίζουν κεντρικό ρόλο σε αυτή την εξέλιξη. Από τους τομείς της Βιομηχανίας 5.0 και της αυτοκίνησης μέχρι τη διασκέδαση, την εκπαίδευση και την υγειονομική περίθαλψη, οι τεχνολογίες αυτές έχουν εισαγάγει καινοτόμες λύσεις που επιτρέπουν σε υπολογιστικά συστήματα να μαθαίνουν από τα περιβαλλοντικά δεδομένα, καθιστώντας τα δυνατά σε πλήρη αυτονομία με ικανότητα λήψης αποφάσεων χωρίς την ανθρώπινη παρέμβαση.

Παραδοσιακά, η Τεχνητή Νοημοσύνη σε περιβάλλοντα δικτύου υλοποιείται με κεντρικοποιημένο τρόπο, με τη συλλογή και επεξεργασία δεδομένων να πραγματοποιούνται σε κεντρικά υπολογιστικά νέφη. Ωστόσο, οι τελευταίες έρευνες έχουν στρέψει την προσοχή τους προς τις κατανεμημένες λύσεις, ώστε να αξιοποιηθούν τα δεδομένα που παράγονται από τις κινητές συσκευές. Σε αντίθεση με την Κεντρικοποιημένη Μάθηση (Centralized Learning - CL), οι μέθοδοι Κατανεμημένης Μάθησης, όπως η Συνεργατική Μάθηση (Federated Learning - FL), μεταφέρουν τον υπολογιστικό φόρτο στις συσκευές, προσφέροντας οφέλη όπως η επεκτασιμότητα, η μείωση κόστους και η προστασία της ιδιωτικότητας των δεδομένων των χρηστών.

Η υπάρχουσα έρευνα για την Κατανεμημένη Μάθηση επικεντρώνεται κυρίως στην απόδοση (ακρίβεια) των εκπαιδευμένων μοντέλων, αγνοώντας συχνά τις πρακτικές πτυχές, όπως η επίδραση στην κατανάλωση πόρων του δικτύου. Η παρούσα διατριβή επιδιώκει να καλύψει αυτά τα κενά ερευνώντας την εφαρμογή των σχημάτων Κατανεμημένης Μάθησης από την πλευρά του συστήματος. Συγκεκριμένα, εξετάζει τις εφαρμογές Αυτόνομης και Διασυνδεδεμένης Κινητικότητας στον τομέα της αυτοκινητοβιομηχανίας, οι οποίες έχουν αυστηρές απαιτήσεις τόσο ως προς την απόδοση του δικτύου (π.χ. καθυστερήσεις) όσο και ως προς την απόδοση των εφαρμογών (π.χ. ασφάλεια).

Ξεκινώντας, πραγματοποιούμε μια συγκριτική αξιολόγηση της απόδοσης μεταξύ Κεντρικοποιημένης και Συνεργατικής Μάθησης, αναλύοντας την αποδοτικότητα της εκπαίδευσης και την κατανάλωση πόρων σε όλο το φάσμα ενός δικτύου: τους χρήστες, το δίκτυο και τις υποδομές στο υπολογιστικό νέφος και την άκρη του δικτύου. Εξετάζουμε το πολύπλοκο πρόβλημα της επιλογής του σχήματος Μηχανικής Μάθησης, λαμβάνοντας υπόψη διάφορες παραμέτρους και περιορισμούς του συστήματος, συμπεριλαμβανομένων και των παραμέτρων δικτύου και κινητικότητας, καθώς και μετρικών Μηχανικής Μάθησης (π.χ. σύγκλιση).

Στη συνέχεια, βασιζόμενοι στις νέες τεχνικές της Συνεχούς Μηχανικής Μάθησης, εξε-

τάζουμε το φαινόμενο της Μετατόπισης Εννοιών, δηλαδή το πώς οι αλλαγές στις κατανομές δεδομένων των χρηστών με την πάροδο του χρόνου, επηρεάζουν την απόδοση των κατανεμημένων μοντέλων μάθησης, ιδίως σε κινητά και δίκτυα οχημάτων. Αυτά τα δίκτυα είναι εκ φύσεως δυναμικά και ταχέως μεταβαλλόμενα, άρα και επιρρεπή στο φαινόμενο της Μετατόπισης Εννοιών. Στη συνέχεια της διερεύνησης, προτείνουμε καινοτόμες τεχνικές για τη διαχείριση της Μετατόπισης Εννοιών με τρόπο αποδοτικό ως προς τους πόρους του δικτύου.

Τέλος, επικυρώνουμε τα θεωρητικά ευρήματά μας μέσω πραγματικών δοκιμών. Αρχικά, διεξάγουμε μια εκστρατεία μετρήσεων μεγάλης κλίμακας για τη συλλογή δεδομένων ποιότητας υπηρεσίας και κινητικότητας χρηστών στο δικτύου, τα οποία στη συνέχεια χρησιμοποιούνται για την υλοποίηση μιας πρακτικής εφαρμογής Συνεργατικής Μάθησης: την κατανεμημένη πρόβλεψη παραμέτρων του δικτύου. Το πλαίσιο μας επεκτείνεται για τη διαχείριση και ενορχήστρωση πολλαπλών υπηρεσιών Συνεργατικής Μάθησης, εισάγοντας έναν ενορχηστρωτή για συσκευές χρηστών που εκτελούν υπηρεσίες Μηχανικής και Συνεργατικής Μάθησης. Ο ενορχηστρωτής αναπτύσσεται σε ένα πραγματικό δίκτυο Πέμπτης Γενιάς και αξιολογείται με κινητές (εντός οχημάτων) και στατικές (εργαστηριακές) συσκευές. Τα αποτελέσματα επιδεικνύουν τη δυνατότητα διαχείρισης του κύκλου ζωής σε πολλαπλές υπηρεσίες, επισημαίνοντας τη δυνατότητα εφαρμογής της Συνεργατικής Μάθησης σε μεγάλης κλίμακας περιβάλλοντα και εφαρμογές Αυτόνομης και Διασυνδεδεμένης Κινητικότητας.

Λέξεις Κλειδιά

Κατανεμημένη Μάθηση, Συνεργατική Μάθηση, Δίκτυα Κινητών Επικοινωνιών, Εφαρμογές Κινητικότητας

Abstract

5G networks and beyond (B5G) are expected to transform mobile communications, enabling the seamless integration of people, devices, and sensors (Internet of Things - IoT) within cyber-physical environments, ultimately realizing the concept of the "Internet-of-Everything" (IoE). A key enabler of this transformation is the convergence of Cloud Computing and the IoT domain, referred to as the 5G network and compute continuum. This continuum facilitates data exchange, processing, and decision-making across all 5G domains, from the cloud and network edge to the IoT domain, known as the Extreme-Edge.

Artificial Intelligence and Machine Learning (AI/ML) play a central role in this evolution. From Industry 5.0 and automotive sectors to infotainment, education, and e-health, AI/ML technologies have introduced innovative solutions that allow computer systems to learn from environmental data, enabling fully autonomous systems capable of decision-making in a human-like manner.

Traditionally, AI/ML in network environments has been implemented in a centralized manner, with data collection and processing occurring in central clouds. However, recent research has shifted focus toward distributed solutions to leverage the data generated by mobile client devices. Unlike Centralized Learning (CL), Distributed Learning (DML) methods, such as Federated Learning (FL), offload computation to client devices, offering benefits such as scalability, cost-efficiency, and privacy preservation for user data.

Existing research on DML primarily focuses on the performance (accuracy) of trained models, often overlooking the practical aspects, such as the impact on underlying network resource consumption. This dissertation seeks to address these gaps by investigating the implementation of DML schemes from a systems perspective. Specifically, it examines the Cooperative, Connected, and Automated Mobility (CCAM) applications in the automotive domain, which have stringent requirements for both network performance (*e.g.,* latency) and application performance (*e.g.,* safety).

To begin, we conduct an end-to-end performance comparison between CL and FL, analyzing training efficiency and resource consumption across all network stakeholders: clients, the network, and cloud/edge infrastructure. We explore the complex issue of ML scheme selection, considering various system parameters and constraints, including network and mobility conditions as well as AI/ML metrics (e.g., convergence). This analysis identifies the trade-offs between critical parameters when choosing between CL and FL.

Next, inspired by ML operations (MLOps) and continuous learning, we examine how concept drift—changes in data distributions over time—affects the performance of distributed ML models, particularly in mobile and vehicular networks like those used in CCAM applications. These networks are highly dynamic and prone to drift. After understanding the impact of concept drift, we propose novel techniques to manage it in a resource-efficient manner.

Finally, we validate our simulation-based findings through real-world testing. We first conduct a large-scale measurement campaign to collect network Quality-of-Service (QoS)

and mobility data, which is then used to demonstrate a practical FL application: distributed QoS prediction. Our framework is extended to manage and orchestrate multiple FL services, introducing an orchestrator for Extreme-Edge and IoT devices performing AI/ML tasks. This orchestrator is deployed on a commercial-grade 5G testbed and evaluated using both mobile (in-vehicle) and static (lab-based) devices. The results demonstrate the feasibility of lifecycle management for multiple services, particularly in the automotive sector, showcasing the potential of FL in large-scale environments.

Keywords

Distributed Machine Learning, Federated Learning, Mobile Network, Vehicular Applications

Contents

List of Figures

List of Tables

Glossary

Abbreviation	English term	Greek term
5G	Fifth Generation (Mobile Networks)	Πέμπτη Γενιά (Κινητά Δίκτυα)
AI	Artificial Intelligence	Τεχνητή Νοημοσύνη
AP	Access Point	Σημείο Πρόσβασης
B5G	Beyond Fifth Generation	Πέρα από την Πέμπτη Γενιά (Κινητά Δίκτυα)
BS	Base Station	Σταθμός Βάσης
CCAM	Cooperative, Connected, and Automated Mobility	Αυτόνομη και Διασυνδεδεμένη Κινητικότητα
CL	Centralized Learning	Κεντρικοποιημένη Μάθηση
DC	Data Center	Κέντρο Δεδομένων
DML	Distributed Machine Learning	Κατανεμημένη Μηχανική Μάθηση
EED	Extreme Edge Device	Συσκευή μετά την Άκρη του Δικτύου (συσκευή χρήστη)
FL	Federated Learning	Συνεργατική Μάθηση
GPU	Graphics Processing Unit	Μονάδα Επεξεργασίας Γραφικών
IoE	Internet of Everything	Υπερδίκτυο των Πραγμάτων
IoT	Internet of Things	Διαδίκτυο των Πραγμάτων
MEC	Mobile Edge Computing	Υπολογιστικοί Πόροι στην Άκρη του Δικτύου
ML	Machine Learning	Μηχανική Μάθηση
MLOps	Machine Learning Operations	Διαδικασίες Μηχανικής Μάθησης
NPU	Neural Processing Unit	Μονάδα Επεξεργασίας Νευρώνων
pQoS	Predictive Quality of Service	Προβλεπόμενη Ποιότητα Υπηρεσίας
QoS	Quality of Service	Ποιότητα Υπηρεσίας
SL	Standard Learning	Τυπική Μάθηση
TPU	Tensor Processing Unit	Μονάδα Επεξεργασίας Τανυστών
UE	User Equipment	Συσκευή Χρήστη

Εκτεταμένη Περίληψη

Οι κινητές τηλεπικοινωνίες υπεισέρχονται σε μια φάση ραγδαίων μεταβολών και μετασχηματισμού με την έλευση των δικτύων Πέμπτης και επόμενης Γενιάς (Beyond 5G - B5G), τα οποία αναμένεται να οδηγήσουν σε πλήρως διασυνδεδεμένα οικοσυστήματα, όπου οι άνθρωποι, οι συσκευές και οι αισθητήρες θα αλληλεπιδρούν μέσα από έξυπνα συστήματα.

Ακρογωνιαίο λίθο προς αυτήν την κατεύθυνση αποτελεί η ενοποίηση της Υπολογιστικής Νέφους (Cloud Computing) με το Διαδίκτυο των Πραγμάτων (Internet of Things - IoT), δημιουργώντας ένα ενιαίο πλαίσιο διασύνδεσης και αξιοποίησης των υπολογιστικών πόρων του δικτύου. Έτσι, η αρχιτεκτονική αυτή επιτρέπει τη ροή δεδομένων, την ανάλυση και τη λήψη αποφάσεων σε πραγματικό χρόνο, σε όλο το εύρος του δικτύου, από το Υπολογιστικό Νέφος και τα Άκρα του δικτύου (Edge), έως τις συσκευές σε επίπεδο χρήστη (Extreme-Edge).

Ενισχυτικά σε αυτήν την τάση λειτουργούν και οι ταχέως αναπτυσσόμενες τεχνολογίες στον τομέα της Τεχνητής Νοημοσύνης (Artificial Intelligence - AI) και της Μηχανικής Μάθησης (Machine Learning - ML). Χάρη στους προηγμένους αλγορίθμους που αναπτύσσονται μέσω των τεχνολογιών αυτών, προωθείται ακόμα πιο έντονα η αυτοματοποίηση, η ψηφιοποίηση, η λήψη αποφάσεων και η αλληλεπίδραση ανθρώπου-μηχανής σε μια σειρά κλάδους, όπως η βιομηχανία, η αυτοκίνηση, οι υπηρεσίες υγείας και εκπαίδευσης, οι μεταφορές, η ψυχαγωγία κ.α.

Εξάλλου οι αρχές που πρεσβεύει η Τεχνητή Νοημοσύνη, δηλαδή η ολοένα και μεγαλύτερη δυνατότητα των υπολογιστικών συστημάτων να μαθαίνουν και να προσαρμόζονται, επιτρέπουν στα συστήματα αυτά να προβλέπουν τις ανάγκες των χρηστών, να βελτιστοποιούν τη χρήση πόρων και να λειτουργούν με πρωτοφανή αυτονομία. Πέρα όμως από τη βελτίωση της απόδοσης των δικτύων, η συνέργεια της Τεχνητής Νοημοσύνης με τα δίκτυα Πέμπτης και επόμενης γενιάς αναμένεται να προωθήσει την εμφάνιση καινοτόμων υπηρεσιών και εφαρμογών, μεταμορφώνοντας θεμελιωδώς τον τρόπο με τον οποίο η κοινωνία αλληλεπιδρά με την τεχνολογία.

Η παρούσα διατριβή έχει ως σκοπό να φωτίσει αυτή ακριβώς τη συνέργεια και να αναλύσει τις καινοτόμες τεχνολογίες της Τεχνητής Νοημοσύνης και της Μηχανικής Μάθησης σε δικτυακά περιβάλλοντα, εστιάζοντας κατά βάση σε εφαρμογές στον τομέα της Αυτόνομης και Διασυνδεδεμένης Κινητικότητας όπως η αυτόνομη οδήγηση, η συνεργατική αντίληψη των οχημάτων κ.α. Για το σκοπό αυτό η διατριβή διαιρείται σε τρεις βασικές θεματικές ενότητες. Στην πρώτη, πραγματοποιείται μια εκτεταμένη ανάλυση των καινοτόμων τεχνικών Κατανεμημένης Μηχανικής Μάθησης σε δικτυακά περιβάλλοντα και σύγκριση αυτών με τις (παραδοσιακές) κεντρικοποιημένες τεχνικές. Έπειτα, εστιάζουμε στη συμπεριφορά των τεχνικών αυτών στο χρόνο, μελετώντας πώς επιδρούν οι αλλαγές των δεδομένων των χρηστών σε τέτοια συστήματα. Τέλος, υλοποιούμε τα θεωρητικά αποτελέσματα ως πρωτότυπες λύσεις σε δικτυακά περιβάλλοντα και εφαρμογές προς εξαγωγή συμπερασμάτων σε πραγματικό επίπεδο χρηστών.

Εισαγωγικές Έννοιες

Συγκεκριμένα ξεκινώντας από την Ενότητα 1, περιγράφουμε τις εισαγωγικές έννοιες, προς διευκόλυνση του αναγνώστη για την περιγραφή των τεχνικών χαρακτηριστικών της Μηχανικής Μάθησης σε περιβάλλοντα δικτύου, που θα ακολουθήσει στις επόμενες ενότητες.

Αρχικά δίνουμε τους βασικούς ορισμούς για τα διάφορα επίπεδα και τις συσκευές ενός δικτύου που αφορούν τις εφαρμογές που μελετάμε. Παρουσιάζουμε τις σύγχρονες εξελίξεις και ιδιαίτερα τις επαναστατικές αλλαγές που επιφέρουν τα Δίκτυα Πέμπτης και επόμενης Γενιάς στις κινητές επικοινωνίες, την επίπτωση αυτών στις συσκευές των χρηστών, αλλά ταυτόχρονα και τις νέες ευκαιρίες που παρουσιάζονται καθώς οι συσκευές αυτές αποκτούν μεγάλη

υπολογιστική ικανότητα. Σε αυτό το πλαίσιο αναλύουμε και την έννοια του Υπολογιστικού Συνεχούς (compute continuum), ως μια συνένωση των υπολογιστικών κόμβων του δικτύου.

Εν συνεχεία προχωράμε με την εισαγωγή των εννοιών της Τεχνητής Νοημοσύνης που θα μας απασχολήσουν στα επόμενα κεφάλαιο. Η Τεχνητή Νοημοσύνη και η Μηχανική Μάθηση έχουν αναδειχθεί ως λύσεις σε μια σειρά από σύνθετα προβλήματα, τα οποία δεν ήταν δυνατό να αντιμετωπιστούν με τις παραδοσιακές αναλυτικές τεχνικές. Για αυτό το λόγο η εφαρμογή αυτών είναι πλέον ευρέως διαδεδομένη π.χ., σε κινητές εφαρμογές, στα αυτόνομα οχήματα, σε περιβάλλοντα έξυπνων αισθητήρων και μικροσυσκευών κ.α.

Σε δικτυακά περιβάλλοντα όπου υπάρχει το στοιχείο της κινητικότητας, μια βασική πρόκληση είναι ότι τα δεδομένα (απαραίτητα για τα μηχανικά μοντέλα) παράγονται στις συσκευές των χρηστών με κατανεμημένο τρόπο, ενώ η αναγκαία (για την εκπαίδευση των μοντέλων) υπολογιστική ισχύς κατά βάση εδράζεται σε κεντρικοποιημένους διακομιστές π.χ., σε υπολογιστικά κέντρα.

Προκειμένου να αρθεί αυτός ο περιορισμός, εφαρμοζόταν παραδοσιακά η λύση της Κεντρικοποιημένης Μάθησης (Centralized Machine Learning - CL). Σε αυτήν την περίπτωση οι χρήστες μεταφέρουν τα δεδομένα τους στον κεντρικό διακομιστή, όπου και εκπαιδεύονται έπειτα τα μοντέλα. Αυτή η λύση ωστόσο αποτελεί μονολιθική προσέγγιση, καθώς αφενός παραβιάζει την ιδιωτικότητα των δεδομένων χρηστών και αφετέρου μπορεί να υπερφορτώσει το δίκτυο, εφόσον απαιτεί συνεχή μεταφορά μεγάλου όγκου δεδομένων.

Για να αντιμετωπιστούν τέτοια ζητήματα, αναδείχθηκαν οι τεχνικές της Κατανεμημένης Μάθησης, με χαρακτηριστικά παραδείγματα την Υβριδική, την Ομότιμη και την Συνεργατική Μάθηση. Η τελευταία, η οποία προέρχεται από την Google, σχεδιάστηκε με σκοπό την βελτίωση της ιδιωτικότητας των χρηστών κατά την εκπαίδευση. Σε ένα σχήμα Συνεργατικής Μάθησης (Federated Machine Learning - FL), ο κεντρικός διακομιστής κατανέμει τα προς εκπαίδευση μοντέλα στους χρήστες, οι οποίοι τα εκπαιδεύουν χρησιμοποιώντας τα δεδομένα τους. Εν συνεχεία, τα επιμέρους μοντέλα των χρηστών συγκεντρώνονται στον κεντρικό διακομιστή, όπου και συνενώνονται σε ένα ανανεωμένο κεντρικό μοντέλο. Η διαδικασία περιλαμβάνει πολλαπλές επαναλήψεις ωσότου επιτευχθεί (ικανοποιητική) σύγκλιση του μοντέλου. Με αυτόν τον τρόπο, τα δεδομένα παραμένουν στον χρήστη σε κάθε περίπτωση και άρα διασφαλίζεται η ιδιωτικότητα.

Κεντρικοποιημένες και Κατανεμημένες τεχνικές Μηχανικής Μάθησης

Στην Ενότητα 2 αναλύουμε τις τεχνικές Κατανεμημένης Μάθησης, συγκρίνοντάς με τις αντίστοιχες (εδραιωμένες) τεχνικές της Κεντρικοποιημένης Μάθησης και αναδεικνύουμε τα πλεονεκτήματα και τα μειονεκτήματα κάθε τεχνικής τόσο από τη σκοπιά της επίδοσης των μοντέλων, αλλά παράλληλα και ως προς αποτύπωμά τους στην κατανάλωση των πόρων του δικτύου, προκειμένου να επιτευχθεί η εκπαίδευση.

Συγκεκριμένα πιάνουμε το νήμα από την προηγούμενη ενότητα, όπου επεξηγήθηκαν οι διαφορετικές τεχνικές Μηχανικής Μάθησης σε δικτυακά περιβάλλοντα και ορίζουμε το πρόβλημα της επιλογής τεχνικής Μηχανικής Μάθησης ως εξής: ποια είναι η βέλτιστη επιλογή σχήματος Μηχανικής Μάθησης (μεταξύ Κεντρικοποιημένης και Συνεργατικής) προκειμένου να εκπαιδευτούν ακριβή μοντέλα, με τους περιορισμούς που θέτει το δίκτυο ως προς την κατανάλωση πόρων.

Αρχικά πραγματοποιούμε αναλυτική βιβλιογραφική ανασκόπηση για το συγκεκριμένο πρόβλημα ως ερευνητικό αντικείμενο. Η ανασκόπηση κατέληξε στο ότι οι έως τώρα ερευνητικές εργασίες που αφορούν τη σύγκριση των τεχνικών Μηχανικής Μάθησης εστιάζουν κατά βάση (είτε με θεωρητικό είτε με πειραματικό τρόπο) στην επίδοση των παραγόμενων μοντέλων, αγνοώντας την επίδραση του κάθε σχήματος στην κατανάλωση των δικτυακών πόρων. Επι-

προσθέτως, οι περισσότερες εργασίες δεν λαμβάνουν υπόψιν την επίδραση των βασικών παραμέτρων του συστήματος στην εκπαίδευση των μοντέλων, όπως για παράδειγμα τις πιθανές διαφοροποιήσεις στις κατανομές των δεδομένων των χρηστών ανά περίπτωση, τη διαθεσιμότητα των χρηστών (η οποία δεν είναι δεδομένη, ιδιαίτερα σε ένα δικτυακό περιβάλλον), τις καθυστερήσεις στην επικοινωνία, την απώλεια πακέτων δεδομένων κ.α.

Δεδομένης της πολυπλοκότητας του προβλήματος λόγω των διαφορετικών απαιτήσεων που θέτουν οι διάφοροι παράγοντες στο δίκτυο π.χ., οι χρήστες, οι διαχειριστές του δικτύου και των υπολογιστικών κόμβων, εστιάζουμε στα ακόλουθα ερωτήματα: 1) Πώς επηρεάζει η επιλογή των υπερ-παραμέτρων της Μηχανικής Μάθησης την κατανάλωση των πόρων στα προς σύγκριση σχήματα Μηχανικής Μάθησης, 2) Πώς επηρεάζει ο λόγος του μεγέθους των δεδομένων προς το μέγεθος του μηχανικού μοντέλου την ακρίβεια των μοντέλων και την κατανάλωση σε κάθε σχήμα, 3) Πώς συμπεριφέρεται κάθε σχήμα στο χρόνο (ταχύτητα σύγκλισης), 4) Ποια η επεκτασιμότητα κάθε σχήματος (κατά την αύξηση των χρηστών) και 5) Πώς επηρεάζει κάθε σχήμα η ανομοιογένεια στα δεδομένα των χρηστών.

Προκειμένου να απαντηθούν τα παραπάνω ερωτήματα, κατασκευάζεται ένα μοντέλο προσομοίωσης δικτύου και υπολογιστικών πόρων, το οποίο περιλαμβάνει όλες τις οντότητες του δικτύου, από τον τελικό χρήστη μέχρι τον κεντρικό διακομιστή. Στοχεύοντας στην προσομοίωση ενός ρεαλιστικού συστήματος, το μοντέλο δικτύου παραμετροποιείται με βάση μετρήσεις από πραγματικά δικτυακά συστήματα και από τις τιμές των κατασκευαστών σε εμπορικές δικτυακές συσκευές. Θεωρούμε δύο χαρακτηριστικές περιπτώσεις για το δίκτυο α) ένα παράδειγμα κινητών επικοινωνιών Long-Term Evolution (LTE) και β) ένα ασύρματο τοπικό δίκτυο τύπου wireless local area (WLAN). Η κινητικότητα των χρηστών για κάθε περίπτωση εισάγεται στο μοντέλο μέσω ιχνών κινητικότητας (mobility traces), από προηγούμενες ερευνητικές εργασίες. Η διαδικασία της εκπαίδευσης στο σύστημα προσομοιώνεται μέσω λογισμικού Κατανεμημένης Μηχανικής Μάθησης, βασισμένο σε γλώσσα προγραμματισμού Python, το οποίο παραθέτουμε στην ερευνητική κοινότητα για την αναπαραγωγή των αποτελεσμάτων, ως λογισμικό ανοιχτού κώδικα (open-source).

Με βάση τις προσομοιώσεις προκύπτουν μια σειρά από αποτελέσματα για τη σύγκριση των δύο σχημάτων Μηχανικής Μάθησης. Αρχικά αναδεικνύεται η ανάγκη παραμετροποίησης των υπερ-παραμέτρων της Μηχανικής Μάθησης ιδιαίτερα για την περίπτωση της Συνεργατικής Μάθησης, προκειμένου να επιτευχθεί ικανοποιητική σύγκλιση. Στη συνέχεια παρουσιάζουμε το πώς ο λόγος των δεδομένων προς το μέγεθος του μοντέλου καθορίζει την κατανάλωση των πόρων του δικτύου καθώς και τα διαστήματα τιμών του λόγου αυτού όπου υπερτερεί το κάθε σχήμα ως προς την ακρίβεια εκπαίδευσης και την κατανάλωση πόρων. Η ανάλυση για όλες τις παραμέτρους του δικτύου παρουσιάζεται στον Πίνακα που ακολουθεί. Αναδεικνύεται επίσης η ικανότητα της Κεντρικοποιημένης Μάθησης να συγκλίνει (έως και 13 φορές) ταχύτερα σε σχέση με τη Συνεργατική, ωστόσο δημιουργώντας τοπικές εκρήσεις (bursts) στην κίνηση δεδομένων του δικτύου. Όσον αφορά την επεκτασιμότητα, η αύξηση των χρηστών επιταχύνει τη σύγκλιση της Συνεργατικής Μάθησης μέχρι και κατά 83%, με αμηλετέο κόστος στην τελική ακρίβεια του παραγόμενου μοντέλου. Τέλος, δείχνουμε την υπεροχή της Κεντρικοποιημένης Μάθησης, όταν στους χρήστες επικρατεί μεγαλός βαθμός ανομοιογένειας δεδομένων, το οποίο αποτελεί πρόκληση στην περίπτωση της Συνεργατικής Μάθησης, με ανάγκη περαιτέρω διερεύνησης για πιθανούς τρόπους αντιμετώπισης σε ένα πραγματικό σύστημα.

Συνεργατική Μάθηση προϊόντος του χρόνου

Εφόσον στην προηγούμενη ενότητα διακρίναμε τα πλεονεκτήματα και τα μειονεκτήματα της Κατανεμημένης έναντι της Κεντρικοποιημένης Μάθησης, στην Ενότητα 3 εστιάζουμε στην περίπτωση της Κατανεμημένης και ιδιαίτερα στην Συνεργατική Μάθηση και εξετάζουμε τη

Επίδραση του λόγου των δεδομένων προς το μέγεθος του μοντέλου r_{data} στην επίδοση και την κατανάλωση πόρων για την Κεντρικοποιημένη (CL) και την Συνεργατική Μάθηση (FL)

Μετρική	CL	FL	Προτεινόμενο σχήμα όταν ($r_{data} \to 0$)	Προτεινόμενο σχήμα όταν ($r_{data} \to \infty$)
Ακρίβεια	Σταθερή	Σταθερή	Κανένα	Κανένα
Κατανάλωση (Εύρος Ζώνης)	Σταθερή	Εκθετική Μείωση	CL	FL
Απώλεια πακέτων	Σταθερή	Γραμμική Μείωση	CL	Κανένα
Συνολική Κατανάλωση Ενέργειας (Χρήστες)	Σταθερή	Γραμμική Αύξηση	CL	CL
Κατανάλωση Ενέργειας ανά Χρήστη	Σταθερή	Γραμμική Αύξηση	CL	CL
Απώλεια Ενέργειας	Σταθερή	Γραμμική Αύξηση	CL	CL
Συνολική Κατανάλωση Ενέργειας (Δίκτυο)	Σταθερή	Εκθετική Μείωση	CL	FL
Συνολική Κατανάλωση Ενέργειας (Νέφος)	Σταθερή	Εκθετική Μείωση	FL	FL

συμπεριφορά αυτής στο χρόνο. Αναδεικνύουμε τις προκλήσεις που προκύπτουν όταν σε ένα πραγματικό σύστημα τα δεδομένα των χρηστών αλλάζουν στο χρόνο, την επίδραση αυτών των αλλαγών στα μοντέλα Συνεργατικής Μάθησης, καθώς και τρόπους αντιμετώπισης τέτοιων φαινομένων.

Κίνητρο για την αναζήτηση αυτή αποτελεί η έννοια της Συνεχούς Μάθησης, η οποία επιτρέπει στα μοντέλα να εκπαιδεύονται περιοδικά και να χρησιμοποιούνται σε βάθος χρόνου. Αυτό συμβαίνει διότι σε ένα πραγματικό σύστημα τα δεδομένα συλλέγονται σταδιακά (σε χρονικό ορίζοντα ημερών, εβδομάδων ή και μηνών), άρα το σύνολο των δεδομένων δεν είναι διαθέσιμο εξαρχής για την εκπαίδευση. Στο πλαίσιο αυτό, προκύπτει και το ζήτημα της Μετατόπισης Εννοιών, δηλαδή της αλλαγής των στατιστικών ιδιοτήτων στις κατανομές των δεδομένων των χρηστών, λόγω φαινομένων εποχικότητας, αλλαγών σε τάσεις και συνήθειες, κτλ.

Η Μετατόπιση Εννοιών ως φαινόμενο μπορεί να επιφέρει μεγάλες μεταπτώσεις στην αποτελεσματικότητα των εκπαιδευμένων μοντέλων αν δεν αντιμετωπιστεί καταλλήλως. Σε κεντρικοποιημένα περιβάλλοντα, όπου υπάρχει πρόσβαση στα δεδομένα των χρηστών το φαινόμενο αυτό έχει μελετηθεί ευρέως και αντιμετωπίζεται μέσω στατιστικών τεχνικών. Στα κατανεμημένα περιβάλλοντα όμως οι κεντρικοποιημένες λύσεις δεν μπορούν να εφαρμοστούν καθώς α) τα δεδομένα των χρηστών προστατεύονται και άρα δεν είναι διαθέσιμα κεντρικά και β) οι συσκευές των χρηστών έχουν περιορισμένη δυνατότητα επεξεργασίας σε σχέση με έναν κεντρικό διακομιστή και άρα δεν μπορούν δεν μπορούν να εφαρμόσουν αποτελεσματικά τις διάφορες τεχνικές αντιμετώπισης της Μετατόπισης Εννοιών. Το βασικά ερωτήματα που καλείται να απαντήσει η παρούσα ενότητα είναι: α) Το πώς μπορεί να εντοπιστεί η Μετατόπιση Εννοιών σε ένα περιβάλλον Συνεργατικής (κατανεμημένης) Μάθησης εγκαίρως και με ικανοποιητική ακρίβεια και β) Το τι μέθοδοι αντιμετώπισης θα ακολουθηθούν για να μην επηρεαστούν τα μοντέλα παρουσία Μετατόπισης Εννοιών, χωρίς παράλληλα η διαδιακασία αυτή να οδηγήσει σε κατασπατάληση πόρων του δικτύου.

Για τα συγκεκριμένα ερωτήματα πραγματοποιήθηκε βιβλιογραφική ανασκόπηση και εντοπίστηκαν τρεις βασικές διαστάσεις/μέθοδοι στην υπάρχουσα βιβλιογραφία για την αντιμετώπιση της Μετατόπισης Εννοιών: η Προσωποποιημένη Μάθηση, η Ασύχρονη Μάθηση και η Συνεχής Μάθηση. Οι δύο πρώτες τεχνικές φαίνεται να αντιμετωπίζουν το φαινόμενο εις βάρος της αύξησης της πολυπλοκότητας του συστήματος, το οποίο τις καθιστά μη εφαρμόσιμες σε

ένα πραγματικό σύστημα μάθησης σε δικτυακό περιβάλλον με χιλιάδες χρήστες-συσκευές. Η Συνεχής Μάθηση από την άλλη μεριά καταφέρνει να μετριάσει το φαινόμενο καταφεύγοντας στη συνεχόμενη εκπαίδευση, το οποίο όπως δείχνουμε και στα αποτελέσματα αργότερα οδηγεί σε κατασπατάληση δικτυακών πόρων όπως ενέργεια, εύρος ζώνης, κ.α.

Έχοντας αναδείξει τις αδυναμίες των προηγούμενων λύσεων, στη συνέχεια παρουσιάζουμε την καινοτόμο πρότασή μας για την αντιμετώπιση του φαινομένου της Μετατόπισης Εννοιών στην Κατανεμημένη Μάθηση (αλγόριθμος DareFL). Ο αλγόριθμος που προτείνουμε είναι σχεδιασμένος για υλοποίηση σε περιβάλλοντα με περιορισμένη πρόσβαση σε υπολογιστικούς πόρους, όπως σε δικτυακές συσκευές ή σε επεξεργαστές σε (αυτόνομα) οχήματα. Διέπεται από τις αρχές της Συνεργατικής Μάθησης σε σχέση με την προστασία της ιδιωτικότητας των δεδομένων των χρηστών και αξιοποιεί ένα μηχανισμό ελέγχου για να ανιχνεύει την ύπαρξη Μετατόπισης Εννοιών, βασιζόμενο (και επεκτείνοντας σε κατανεμημένα περιβάλλοντα) τον γνωστό κεντρικοποιημένο αλγόριθμο Drift Detection Method (DDM). Εφόσον ανιχνευτεί η Μετατόπιση Εννοιών, αυτή αντιμετωπίζεται μέσω μετ-εκπαίδευσης των μοντέλων (μέσω Συνεργατικής Μάθησης), της οποίας όμως η διάρκεια καθορίζεται αυστηρά με βάση τη σύγκλιση, ώστε να ελαττωθεί η συνολική κατανάλωση πόρων.

Προκειμένου να αξιολογηθεί ο προτεινόμενος αλγόριθμος, μελετάμε ως πρόβλημα Μηχανικής Μάθησης την περίπτωση της πρόβλεψης χρονοσειρών δικτυακών παραμέτρων π.χ., ρυθμός μετάδοσης (predictive Quality of Service - pQoS) από το χώρο της Αυτόνομης και Διασυνδεδεμένης Κινητικότητας. Οι εφαρμογές των οχημάτων βασίζονται στη δικτυακή υποδομή προκειμένου να ανταλλάζουν δεδομένα και κρίσιμα μηνύματα, όπως για παράδειγμα στην περίπτωση της αυτόνομης οδήγησης. Σε περίπτωση που το δίκτυο δεν μπορεί να καλύψει τα αυστηρά κριτήρια που θέτουν οι εφαρμογές αυτές π.χ., ιδιαίτερα χαμηλές καθυστερήσεις πακέτων, εγκυμονούνται κίνδυνοι ως προς την ορθή λειτουργία των εφαρμογών αυτών και τελικά την ασφάλεια του χρήστη-οδηγού. Στην περίπτωση του pQoS το δίκτυο έχει τη δυνατότητα να προβλέψει τέτοιες καταστάσεις στο εγγύς μέλλον και να ειδοποιήσει τις εφαρμογές, οι οποίες με τη σειρά τους μπορούν να προσαρμόσουν τη λειτουργία τους με ασφαλή για το χρήστη τρόπο π.χ., να μειώσουν την ταχύτητα του οχήματος ή να δώσουν τον έλεγχο ενός αυτόνομου οχήματος στον χρήστη.

Με αυτήν την έννοια η πρόβλεψη των δικτυακών παραμέτρων αποτελεί βασικό εργαλείο του τομέα της Αυτόνομης και Διασυνδεδεμένης Κινητικότητας, ενώ παράλληλα πρόκειται για χαρακτηριστική περίπτωση όπου τα δεδομένα των χρηστών (σχετικά με το δίκτυο, την κινητικότητα των χρηστών, κα.) μεταβάλλονται με ραγδαίους ρυθμούς και απροσδιόριστους τρόπους και άρα το φαινόμενο της Μετατόπισης Εννοιών έχει συνήθως μεγάλη ένταση και συχνότητα, συμπέρασμα στο οποίο καταλήγει και η υπάρχουσα βιβλιογραφία.

Δεδομένης της έλλειψης pQoS δεδομένων με παραδείγματα Μετατόπισης Εννοιών σε δίκτυα, επικεντρωνόμαστε στην παραγωγή τέτοιων δεδομένων μέσω λογισμικού παραγωγής συνθετικών δικτυακών δεδομένων (OMNET++), το οποίο ενσωματώνει αντίστοιχο λογισμικό κινητικότητας οχημάτων σε αστικό περιβάλλον (SUMO). Συγκεκριμένα μελετώνται δύο σενάρια Μετατόπισης Εννοιών. Το πρώτο αναφέρεται σε αλλαγές στην υποδομή του δικτύου (μείωση του αριθμού των σταθμών βάσης στην περιοχή εξυπηρέτησης των χρηστών), ενώ το δεύτερο σε αλλαγές στην κινητικότητα των χρηστών-οχημάτων (μεταβολή των διαδρομών των οχημάτων στην περιοχή κάλυψης).

Με αυτόν τον τρόπο επιτυγχάνουμε ρεαλιστική αναπαράσταση φαινομένων Μετατόπισης Εννοιών στο δίκτυο και εξαγωγή των αντίστοιχων πακέτων δεδομένων προς επεξεργασία για την περίπτωση του pQoS. Εν συνεχεία τα δεδομένα εισάγονται στον προσομοιωτή Κατανεμημένης Μάθησης που περιγράφηκε στην προηγούμενη ενότητα, όπου και συγκρίνουμε τον προτεινόμενο αλγόριθμο Συνεργατικής Μάθησης παρουσία Μετατόπισης Εννοιών με άλλες τρεις υπάρχουσες λύσεις στη βιβλιογραφία. Τόσο τα δεδομένα που παρήχθησαν όσο και το

λογισμικό (Συνεχούς) Συνεργατικής Μάθησης για την περίπτωση του pQoS διατίθενται στην ερευνητική κοινότητα μέσω αποθετηρίου.

Στα αποτελέσματα που παρουσιάζουμε σε αυτήν την ενότητα, δείχνουμε ότι ο προτεινόμενος αλγόριθμος επιτυγχάνει έγκαιρη διάγνωση της Μετατόπισης Εννοιών και αντιμετώπιση αυτής εν συνεχεία. Συγκεκριμένα, η τελική ακρίβεια των μοντέλων είναι παρόμοια με αυτήν των υπόλοιπων λύσεων (Συνεχούς Μάθησης) με διαφορές που κυμαίνονται εντός του ορίου του 10%, για τα δύο σενάρια της προσομοίωσης. Παράλληλα, παρουσιάζουμε πώς ο αλγόριθμος καταφέρνει να μειώσει τα κόστη επικοινωνίας (communication costs) για το δίκτυο (έως και 76%), αλλά και ενεργειακής κατανάλωσης για τον κεντρικό διακομιστή (έως και 74%) και τους χρήστες (έως και 68%), σε σχέση με τις υπάρχουσες λύσεις.

Τέλος, εξετάζουμε αποκλειστικά την ικανότητα ανίχνευσης της Μετατόπισης Εννοιών του προτεινόμενου αλγόριθμου, ο οποίος επιτυγχάνει καλύτερη συνολική αποδόση έως και 9% σε σχέση με τις υπάρχουσες λύσεις, καταναλώνοντας ωστόσο μέχρι και τρεις φορές λιγότερη ενέργεια, ως αποτέλεσμα του σχετικά απλού τρόπου λειτουργίας της μεθόδου Drift Detection Method (DDM) σε σχέση με τις υπάρχουσες λύσεις σύγκρισης των παραμέτρων των μηχανικών μοντέλων. Όλοι αυτοί οι παράγοντες καθιστούν τελικά τον αλγόριθμο DareFL ισάξιο σε επίδοση αλλά αποδοτικότερη σε κόστος λύση, σε σχέση με τις υπάρχουσες μεθόδους.

Πρακτικές εφαρμογές Συνεργατικής Μάθησης

Παίρνοντας τη σκυτάλη από την προηγούμενη ενότητα που μελετήσαμε τη συμπεριφορά της Συνεργατικής Μάθησης σε πραγματικά περιβάλλοντα τα οποία μεταβάλλονται στο χρόνο, στην Ενότητα 4, προχωρούμε στην υλοποίηση των θεωρητικών αποτελεσμάτων που αναλύθηκαν στην προηγούμενη ενότητα, επικεντρώνοντας την προσοχή στην εφαρμογή τους σε πραγματικά συστήματα, τα οποία περιλαμβάνουν μεγάλης κλίμακας δικτυακά περιβάλλοντα και κινητές συσκευές με ικανότητα επεξεργασίας δεδομένων.

Συγκεκριμένα, εστιάζουμε στην εφαρμογή της Συνεργατικής Μάθησης για την πρόβλεψη παραμέτρων ποιότητας υπηρεσίας δικτύου (predictive Quality of Service - pQoS), όπως αναλύθηκε στο προηγούμενο κεφάλαιο, και εξετάζουμε την υλοποίησή της σε περιβάλλοντα Αυτόνομης και Διασυνδεδεμένης Κινητικότητας. Η μελέτη αυτή ξεκινά με τη συλλογή πραγματικών δεδομένων QoS, καθώς τέτοια δεδομένα δεν είναι διαθέσιμα στην υπάρχουσα βιβλιογραφία. Για να καλύψουμε αυτό το κενό, δημιουργήσαμε το σύνολο δεδομένων *NordicDat*, το οποίο περιλαμβάνει πραγματικά δεδομένα ποιότητας υπηρεσίας που συλλέχθηκαν κατά τη διάρκεια μιας εκστρατείας μέτρησης σε τρεις ευρωπαϊκές χώρες: Φινλανδία, Σουηδία και Νορβηγία. Το *NordicDat* περιλαμβάνει 25 ώρες δεδομένων οδήγησης (με διαφορετικές ταχύτητες) και καταγράφει τόσο φυσικά χαρακτηριστικά όσο και χαρακτηριστικά του δικτύου, καθώς και κινηματικά δεδομένα των οχημάτων. Οι μετρήσεις πραγματοποιήθηκαν κοντά σε εθνικά σύνορα, προκειμένου να καταγραφεί η επίδραση της περιαγωγής (roaming) στην ποιότητα υπηρεσίας. Αξιοσημείωτο επίσης είναι το γεγονός ότι το σύνολο δεδομένων περιλαμβάνει καταγραφές από τεχνολογίες που αφορούν δίκτυα μετάδοσης Τέταρτης και Πέμπτης γενιάς.

Αρχικά πραγματοποιήθηκε στατιστική ανάλυση των δεδομένων προκειμένου να εξαχθούν συμπεράσματα ως προς τις σχέσεις μεταξύ των παραμέτρων που καταγράφηκαν. Η ανάλυση των δεδομένων αυτών αποκάλυψε ότι ενώ δεν υπάρχει ισχυρή γραμμική σχέση μεταξύ των παραμέτρων, ο ρυθμός μετάδοσης καθόδου (Downlink Throughput) επηρεάζεται κυρίως από την κινητικότητα των χρηστών, ενώ αντίστοιχα στην άνοδο (Uplink) καθοριστικό ρόλο παίζουν οι παράμετροι στο φυσικό επίπεδο (physical layer) του δικτύου. Στο πεδίο του χρόνου, η παράμετρος της καθυστέρησης (delay) παρουσιάζει χαμηλή αυτοσυσχέτιση, σε αντίθεση με τον ρυθμό μετάδοσης που παρουσιάζει υψηλή (μεγαλύτερη του 0.8) για διάστημα μέχρι και 50 δευτερόλεπτα. Τέλος παρατηρήθηκε πώς οι αλλαγές στην περιαγωγή, στα προφίλ ταχύτητας

και τις τεχνολογίες δικτύου επηρεάζουν σημαντικά τις τιμές του μέσου ρυθμού μετάδοσης.

Το *NordicDat* χρησιμοποιήθηκε για την επίδειξη της πρόβλεψης δικτυακών παραμέτρων μέσω Συνεργατικής Μάθησης. Στο βαθμό που γνωρίζουμε, αυτή είναι η πρώτη απόπειρα να εξερευνηθεί η κατανεμημένη πρόβλεψη δικτυακών παραμέτρων με βάση δημόσια δεδομένα. Το σύνολο δεδομένων *NordicDat*, συνοδευόμενο από λεπτομερή τεκμηρίωση, είναι διαθέσιμο δημόσια σε ανοιχτό αποθετήριο, παρέχοντας μια πολύτιμη πηγή για μελλοντική έρευνα στον τομέα της πρόβλεψης ποιοτικών και ποσοτικών παραμέτρων του δικτύου και καλύπτοντας κρίσιμα κενά στη βιβλιογραφία της Μηχανικής και της Συνεργατικής Μάθησης.

Για τους σκοπούς της εκπαίδευσης κατασκευάστηκε μηχανικό μοντέλο πρόβλεψης Συνεργατικής Μάθησης με ορίζοντα πρόβλεψης έως και οκτώ δευτερόλεπτα, το οποίο αξιολογήθηκε με βάση την επίδοση πρόβλεψης στην καθυστέρηση των πακέτων και τον ρυθμό μετάδοσης. Το μοντέλο της Συνεργατικής Μάθησης κατάφερε να πετύχει επιδόσεις παρόμοιες με τα κλασσικά μοντέλα της Κεντρικοποιημένης Μάθησης με μέσο βαθμό απόκλισης στο 9%, κάτι το οποίο αναδεικνύει την ικανότητα της Συνεργατικής Μάθησης στην εκπαίδευση πραγματικών δεδομένων με την παράλληλη εξασφάλιση της ιδιωτικότητας των δεδομένων.

Εν συνεχεία προχωρήσαμε σε εγκατάσταση κινητών συσκεών σε οχήματα και εφαρμογή της Συνεργατικής Μάθησης σε πειραματικό δικτυακό περιβάλλον. Υιοθετήσαμε μια προσέγγιση προσανατολισμένη σε συστημικό επίπεδο, σχεδιάζοντας και υλοποιώντας έναν ενορχηστρωτή κινητών συσκεών (Extreme-Edge Orchestrator - EEO), δηλαδή ένα πλαίσιο διαχείρισης που ενσωματώνει τους πόρους των κινητών συσκευών στο οικοσύστημα του 5G. Η λύση μας αξιοποιεί εργαλεία του Υπολογιστικού Νέφους (cloud-native) για την παρακολούθηση (monitoring) των πόρων του δικτύου και την διαχείριση του κύκλου ζωής των υπηρεσιών δικτύου (service life-cycle management) που εκτελούνται σε κινητές συσκευές. Οι αποφάσεις που σχετίζονται με τον κύκλο ζωής των υπηρεσιών, όπως η εκκίνηση μιας νέας υπηρεσίας, η επιλογή των συσκευών για την εκτέλεση της υπηρεσίας και ο ομαλός τερματισμός της υπηρεσίας, καθοδηγούνται από πολιτικές που ορίζονται από τον χρήστη, δημιουργώντας μηχανισμούς ελέγχου κλειστού βρόχου. Αυτοί οι βρόχοι ενσωματώνουν διάφορα κριτήρια τα οποία μπορεί να ορίσει ο χρήστης και περιλαμβάνουν: α) χαρακτηριστικά συσκευής (π.χ., επεξεργαστική ικανότητα), β) παραμέτρους κατανάλωσης πόρων δικτύου (π.χ., ενεργειακά κόστη) και γ) χαρακτηριστικά σε επίπεδο εφαρμογής (π.χ., διαθεσιμότητα δεδομένων), που μπορεί να περιλαμβάνουν ακόμα και πληροφορίες σχετικές με την εκπαίδευση μοντέλων Μηχανικής και Συνεργατικής Μάθησης.

Σε αντίθεση με τη συντριπτική πλειοψηφία των ερευνών στον τομέα αυτόν που βασίζονται σε προσομοιώσεις, η προτεινόμενη λύση μας αναπτύσσεται σε ένα λειτουργικό Δίκτυο Πέμπτης Γενιάς, και αξιολογείται μέσω εκτεταμένων πειραματικών μελετών στον πραγματικό κόσμο, που περιλαμβάνουν τόσο κινητές όσο και στατικές υπολογιστικές συσκευές σε ποικιλία πειραματικών σεναρίων (εσωτερικά σε εργαστηριακό επίπεδο και εξωτερικά σε οδικό δίκτυο). Η συνεισφορά μας είναι διπλή: 1) εισάγουμε, σχεδιάζουμε και αναπτύσσουμε μια καινοτόμο λύση για την ορχήστρωση κινητών συσκεών, η οποία μαζί με την αντίστοιχη αρχιτεκτονική συστήματος επεκτείνει τη δυνατότητα υπολογισμού και επεξεργασίας δεδομένων στις παρυφές του Δικτύου (Extreme-Edge), και 2) αξιολογούμε τη λύση μας σε ένα πειραματικό περιβάλλον 5G για την υλοποίηση κατανεμημένης πρόβλεψης δικτυακών παραμέτρων μέσω Συνεργατικής Μάθησης, μια βασική εφαρμογή στον τομέα της Αυτόνομης και Διασυνδεδεμένης Κινητικότητας, όπως αναδείξαμε και παραπάνω.

Τα πειραματικά αποτελέσματα έδειξαν ότι η εισαγωγή του ενορχηστρωτή EEO βελτιώνει την αποδοτικότητα των πόρων μέσω της βελτιστοποίησης βασικών παραμέτρων, όπως η διάρκεια των υπηρεσιών, και μειώνει τον χρόνο ολοκλήρωσης των υπηρεσιών κατά 25% υπό συνθήκες υψηλού φόρτου (σε επίπεδο υπολογιστικής ικανότητας είτε στην μνήμη των συσκευών), σε σύγκριση με τις υπάρχουσες μεθόδους. Η ανάλυσή μας αποδεικνύει την αποτελε-

σματικότητα του μηχανισμού επιλογής συσκευών με πολλαπλά κριτήρια σε σενάρια πολλαπλών υπηρεσιών, όπου ο υπολογιστικός φόρτος στις συσκευές αυξάνεται δραματικά, ιδιαίτερα όταν αναφερόμαστε σε σύγχρονες εφαρμογές που περιλαμβάνουν αλγόριθμους Μηχανικής Μάθησης. Επίσης τα ευρήματα αυτά υποδεικνύουν ότι η διαχείριση συσκευών μέσω ενορχηστρωτή σε δίκτυα επόμενης γενιάς γενικά μπορεί να οδηγήσει σε πιο αποδοτική και αποτελεσματική παροχή υπηρεσιών.

Συμπεράσματα

Τέλος στην Ενότητα 5 συνοψίζουμε την ερευνητική εργασία που παρουσιάστηκε στις προηγούμενες ενότητες της διατριβής, με θέμα την εφαρμογή της Συνεργατικής Μάθησης σε δικτυακά περιβάλλοντα. Η ερευνητική εργασία που παρουσιάστηκε είχε ως στόχο την μελέτη τριών βασικών ερευνητικών κατευθύνσεων: α) το πρόβλημα επιλογής σχήματος Μάθησης, ανάμεσα σε Συνεργατική και Κεντρικοποιημένη από τη σκοπιά της κατανάλωσης δικτυακών πόρων και β) το ρόλο των βασικών παραμέτρων/ιδιοτήτων του συστήματος π.χ., επιλογή κόμβου συνένωσης μοντέλων, ενεργειακή κατανάλωση και το ζήτημα της Μετατόπισης Εννοιών στα μηχανικά μοντέλα. και γ) τις προκλήσεις στην εφαρμογή των θεωρητικών αποτελεσμάτων σε πραγματικά συστήματα για εφαρμογές Αυτόνομης και Διασυνδεδεμένης Κινητικότητας.

Προκειμένου να απαντηθούν τα ερωτήματα αυτά, παρουσιάσαμε το θεωρητικό υπόβαθρο, το περιβάλλον προσομοίωσης καθώς και τα αποτελέσματα που προέκυψαν από αυτό, τα οποία ανέδειξαν τις παρακάτω πλευρές: α) τις παραμέτρους του συστήματος που καθορίζουν την απόδοση κάθε σχήματος Μηχανικής Μάθησης σε σχέση τόσο με την επίδοση (ακρίβεια μοντέλων) όσο και την αντίστοιχη κατανάλωση σε δικτυακούς πόρους, β) το πώς επηρεάζει η τοπολογία του δικτύου και η επιλογή των παραμέτρων της Μηχανικής Μάθησης την ενεργειακή κατανάλωση και τέλος γ) τρόπους αντιμετώπισης της Μετατόπισης Εννοιών σε περιβάλλοντα περιορισμένης υπολογιστικής/ενεργειακής ισχύος.

Εν συνεχεία, παρουσιάσαμε την υλοποίηση των θεωρητικών αποτελεσμάτων σε πραγματικά συστήματα Αυτόνομης και Διασυνδεδεμένης Κινητικότητας. Συγκεκριμένα, αναδείξαμε τις προκλήσεις αλλά και τα αποτελέσματα που προέκυψαν από την εκστρατεία συλλογής δεδομένων δικτύου σε ένα οδικό δίκτυο, καθώς και το πώς μπορούν αυτά να χρησιμοποιηθούν στο πλαίσιο εφαρμογών της Συνεργατικής Μάθησης. Έπειτα παρουσιάσαμε την υλοποίηση ενός ενορχηστρωτή πολλαπλών υπηρεσιών Μηχανικής και Συνεργατικής Μάθησης σε κινητές συσκευές, ο οποίος ενσωματώθηκε σε ένα περιβάλλον διαχείρησης συσκεύων πάνω από πραγματικό δίκτυο Πέμπτης Γενιάς προκειμένου να επικυρωθούν τα αποτελέσματά του πειραματικά.

Τόσο τα θεωρητικά αποτελέσματα που αναφέρθηκαν όσο και οι πρακτικές υλοποιήσεις μπορούν να διευρυνθούν σε διάφορες ερευνητικές κατευθύνσεις. Ιδιαίτερο ενδιαφέρον παρουσιάζει η συνένωση των αποτελεσμάτων επιλογής σχήματος Μηχανικής Μάθησης με τους αλγόριθμους που αντιμετωπίζουν την Μετατόπιση Εννοιών, κάτι το οποίο θα μπορούσε να συνθέσει ένα ολοκληρωμένο σύστημα Συνεχούς Μάθησης, που θα προσαρμόζεται στις αλλαγές των συνηθειών των χρηστών αλλά ταυτόχρονα θα μεταβάλλεται προκειμένου να επιτύχει πιο αποτελεσματική λειτουργία (π.χ., μείωση της κατανάλωσης των δικτυακών πόρων).

1

Foundations

The foundation of this research lies in two interconnected and transformative concepts shaping the landscape of next-generation networks and intelligent systems. The first is the emergence of a unified Compute Continuum, which integrates multiple computational layers within the network, extending from centralized cloud infrastructures to the core network and all the way to edge devices, including handheld and Internet-of-Things (IoT) devices. This continuum provides a seamless framework for data processing and decision-making, enabling complex computations to be executed closer to the data source, thereby reducing latency and enhancing efficiency. By leveraging this architectural evolution, networks are increasingly capable of supporting demanding applications in dynamic environments.

The second concept is the rise of distributed Artificial Intelligence and Machine Learning (AI/ML), a paradigm shift that has gained momentum due to two critical advancements. The first driver is the exponential growth of user-generated data in the Big Data era, fueled by billions of interconnected devices producing vast volumes of heterogeneous data. The second driver is the significant enhancement in computational capabilities, with cutting-edge hardware now deployed closer to the end-user, including edge servers, mobile devices, and IoT nodes. This decentralization of computational resources has unlocked new possibilities for deploying AI/ML algorithms at scale, enabling real-time decision-making and intelligent behavior in distributed systems.

Together, these developments represent a convergence of computational and intelligence paradigms, addressing the increasing demands of applications that require low-latency responses, energy efficiency, scalability, and data privacy. For instance, scenarios such as autonomous driving, smart cities, and real-time industrial automation benefit immensely from these advancements by leveraging the synergy between distributed computing and AI/ML.

This thesis explores the intersection of these concepts, focusing on their application within the demanding and dynamic domain of Cooperative, Connected, and Automated Mobility (CCAM). This vertical exemplifies the challenges and opportunities of the Compute Continuum and distributed AI/ML, with stringent requirements for real-time processing, network efficiency, and safety-critical operations.

In this context, the research addresses fundamental questions: How can distributed AI/ML frameworks, such as Federated Learning (FL), leverage the Compute Continuum to optimize resource utilization while maintaining high accuracy and robustness? What are the trade-offs involved in deploying centralized versus distributed AI/ML schemes, particularly in terms of network and computational resource efficiency? How do real-world constraints, such as mobility, network volatility, and data drift, impact the performance

and reliability of distributed AI/ML?

By examining these questions, this thesis provides a systematic investigation into the theoretical and practical challenges of integrating distributed AI/ML with advanced network architectures. This introductory chapter establishes the groundwork for the analysis, methodologies, and findings presented in the subsequent chapters, offering insights into the transformative potential of these technologies for next-generation intelligent systems.

1.1 Next-Generation networks landscape

Over the past few decades, since the introduction of mobile networks, there has been a profound transformation in both network technologies and the devices and applications associated with them. From the First Generation (1G) networks in the early 1980s, which offered only analog cellular services, to the Fifth Generation and Beyond (B5G) networks, which aim for the full automation of device and human interconnections, the evolution has been substantial. A critical aspect of this progress has been the role of user equipment (UE), serving as the bridge between the physical and digital worlds. This concept was first introduced in 1999 with the term **Internet of Things (IoT)** [1], which describes an interconnected network of physical devices equipped with sensors and capable of data exchange.

Today, IoT is associated with a wide range of devices, from laptops, smartphones, and wearables for infotainment and e-health, to on-board units (OBUs), smart sensors, and collaborative robots (cobots) in industries such as automotive and Industry 5.0. These devices collaborate to create adaptive environments that sense, analyze, communicate, and take actions to meet human needs. As a result, IoT leverages real-time data to transform various aspects of daily life, advancing us toward a smarter and more interconnected future. In addition, advances in mobile networks have expanded this concept to include ubiquitous and autonomous object networks, which emphasize seamless identification and service integration. In this context, Cisco has coined the term **Internet of Everything (IoE)** to describe the full interconnection of people, devices, and environments, enabling the exchange of data and services among them and with other entities [2].

To support such functionalities, IoT networks generate vast amounts of data, commonly referred to as **Big Data** [3]. Data exchange between IoT devices and other entities typically occurs through wireless technologies, such as the Institute of Electrical and Electronics Engineers (IEEE) 802.11 family of standards (Wireless Fidelity - WiFi) [4], the International Telecommunication Union (ITU) ITU-T Y.4480 (Long Range Wide Area Network - LoRaWAN) [5], and cellular networks, including Long-Term Evolution (LTE), 5G, and B5G [6]. Once the communication link is established, the management of the computational and storage needs of Big Data is traditionally handled through **cloud computing technologies**. This approach centralizes data processing in large-scale data centers (DCs), providing near-unlimited storage, computational power, and resources, while facilitating the emergence of innovative service models. Cloud computing operates through a variety of service models, which include: a) Infrastructure as a Service (IaaS), which grants on-demand access to computing, storage, and networking resources, allowing users to scale and manage their infrastructure needs without investing in physical hardware; b) Platform as a Service (PaaS), which provides a comprehensive environment for application development, hosting, and management, simplifying software deployment and reducing the complexity of underlying infrastructure; and c) Software as a Service (SaaS), which enables users to access cloud-based applications, such as email, messaging, and text editing, over the Internet, eliminating the need for local installations and updates while

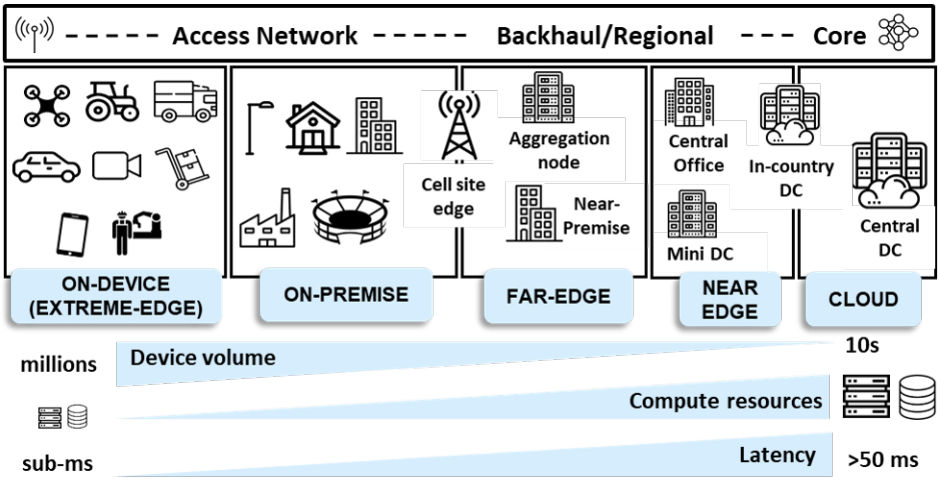


Figure 1.1: The compute continuum

promoting seamless accessibility and collaboration across devices.

Although cloud computing offers significant advantages, it also presents challenges for latency-sensitive applications and services that require high-volume data transmission. Moreover, concerns regarding data privacy and trust arise when relying on third-party-operated cloud servers [3]. To overcome these limitations, the **Mobile Edge Computing (MEC)** paradigm has been introduced. MEC brings computational and storage resources closer to UEs by deploying applications at the network edge [7]. This architecture utilizes the existing mobile network operator (MNO) infrastructure or newly virtualized setups, significantly reducing latency and improving service quality. MEC also enables collaboration between edge and cloud data centers, fostering decentralized yet coordinated ecosystems. This hybrid model is essential for supporting ultra-reliable, low-latency applications, such as immersive augmented/virtual reality (AR/VR) and vehicle-to-everything (V2X) services [8].

In parallel, UEs have evolved into powerful computational and communication nodes, fueled by advancements in hardware accelerators, including Field Programmable Gate Arrays (FPGAs), Application-Specific Integrated Circuits (ASICs), Central Processing Units (CPUs), and Graphics Processing Units (GPUs) [9]. Specialized processors, such as Google's Tensor Processing Units (TPUs) [10] and Qualcomm's Neural Processing Units (NPUs) [11], further enhance computational power, enabling low-power acceleration for application-specific tasks. This evolution has led to the development of **on-device (or Extreme-Edge) computing**, where computational tasks are offloaded to user devices rather than edge servers. This strategy reduces reliance on centralized infrastructure and supports localized data processing [12].

The growing integration of cloud, edge, and on-device computing has given rise to the concept of the **compute continuum**. This paradigm envisions a seamless environment in which computing and network resources across endpoints, edge nodes, and cloud DCs are dynamically coordinated to optimize workload distribution [13]. The compute continuum marks a significant shift toward the convergence of traditionally separate domains, enabling efficient management and security of services and data across various layers; from devices in the access network to edge nodes in the backhaul and cloud DCs in the core network. Each layer offers unique capabilities in terms of computational power, network latency, and heterogeneity (see Fig. 1.1) [6]. The interplay between these layers forms the

foundation for innovative network infrastructures, software architectures and deployment of applications in next-generation networks.

1.2 The role of Artificial Intelligence

Recent advancements in Artificial Intelligence and Machine Learning (AI/ML) have revealed substantial potential for solving complex problems where traditional methods, such as analytical solutions and approximation techniques, either perform inadequately or fail entirely. ML techniques offer data-driven solutions, eliminating the need for manual programming. By leveraging large amounts of historical data, these techniques can identify patterns, analyze behaviors, and even predict future outcomes [14]. As a result, AI/ML technologies are becoming essential components of modern applications, including computer vision, natural language processing (NLP), pattern recognition, *etc.* These applications extend beyond local or private environments, often spanning diverse domains such as mobile applications, autonomous vehicles, IoT, *etc.*

In a broader context, AI refers to the simulation of human intelligence processes, such as learning, problem-solving, and decision-making, by computer systems. ML, a subfield of AI, allows computer systems to learn from data using algorithms without the need for explicit programming. ML algorithms are typically categorized into three main types: supervised learning, unsupervised learning and reinforcement learning.

Supervised learning is a type of ML where a model is trained using labeled data. Each instance in the dataset consists of inputs (features) and corresponding outputs (labels or targets). The objective is for the model to learn a mapping from inputs to outputs so that it can predict the output for new, unseen inputs. Supervised learning problems are typically classified into two categories: classification, which predicts discrete categories *e.g.,* spam email detection, and regression, which predicts continuous values *e.g.,* house price prediction. Common applications include image recognition, NLP, fraud detection, and others. Relevant algorithms include Linear Regression, Support Vector Machines (SVMs), Neural Networks (NN), Deep Neural Networks (DNN), Decision Trees, and Random Forests.

Unsupervised learning refers to a type of machine learning where the model analyzes and clusters unlabeled data. These algorithms identify hidden patterns or groupings in the data without requiring human intervention. The goal of unsupervised learning is to explore the data, detect patterns, or group similar data points. Common applications include anomaly detection, recommendation systems, and customer/market segmentation. Relevant algorithms include K-Means Clustering, Principal Component Analysis (PCA) and Autoencoders. Finally, Reinforcement Learning involves an agent learning to make decisions by interacting with an environment. The agent's objective is to maximize cumulative rewards over time by selecting actions that lead to desired outcomes. Typical applications include gaming, finance (*e.g.,* risk assessment, trading), decision-making, and automation.

Among the various models, Neural Networks (NN), especially Deep Neural Networks (DNN), have attracted considerable attention due to their exceptional performance across diverse fields and applications, including image recognition and large language models (LLMs) [15], [16]. NNs are ML models consisting of interconnected mathematical functions (neurons), with the connections between neurons (called weights) represented as numerical values, as depicted in Fig. 1.2. A typical NN model includes several input and output variables. The model aims to identify the relationship between input and output data, which is based on Kolmogorov's theorem, stating that a continuous multivariate function

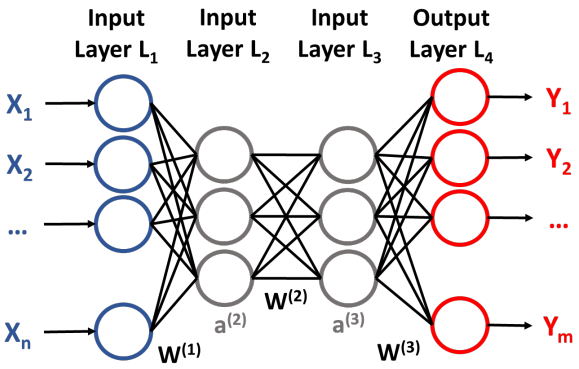


Figure 1.2: A neural network model's architecture

can be expressed on a compact set in terms of sums and compositions of a finite number of single variable functions [17].

To achieve this, the model undergoes a **training phase** where it is fed with a large dataset of known input-output pairs. Through a repetitive process known as back propagation, the model adjusts its weights iteratively to better capture the relationship between input and output variables. The convergence of this process is monitored by comparing the model's predictions with the actual results, using a metric called the loss function, with the objective of minimizing this loss. Once trained, the model enters the **inference (or testing) phase**, where it can make predictions on unseen data, with the expectation that its outputs will closely align with the true values.

To fully harness the potential of AI/ML, data plays a crucial role, both in effectively training the models and utilizing them (for inference) in the end-user domain. Towards that goal, a significant challenge emerges. Client IoT devices that are the main data sources are by nature distributed, mobile and often unreliable, as discussed in Sec. 1.1. In contrast, the processing capacity required for model training typically resides in centralized entities like cloud DCs. Consequently, for any ML pipeline to be implemented, there is a need for a reliable network connection between clients and servers for data exchange as well as an orchestration layer for the learning process that would determine essential factors of the learning process, such as where and when processing occurs and which clients participate.

Traditional approaches to these challenges often rely on Centralized Learning (CL) schemes, as illustrated in Fig. 1.3. In CL, clients periodically upload their data to a central entity, such as a cloud server, which handles the computationally intensive ML training. Once training is complete, a global ML model is produced and distributed back to the clients for inference. This process can be repeated multiple times, using different datasets, until the model converges to a predefined level of performance.

Addressing concerns over user data privacy, Distributed ML (DML) schemes have emerged as alternatives to CL. Notable examples include hybrid approaches [18] and peer-to-peer (server-less) learning frameworks [19], where computational tasks are offloaded to client user equipment (UEs). Among these, Federated Learning (FL), a Google-initiated DML scheme [20], has gained prominence due to its inherent privacy-preserving design. In FL, the training process is decentralized, as depicted in Fig. 1.4. Training occurs in iterative cycles (rounds), during which a central server distributes a global ML model to a subset of clients. These clients locally train the model using their private data and computing resources, then return updated local models to the server. The server

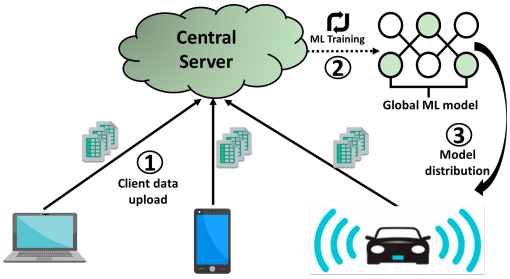


Figure 1.3: Centralized Learning (CL)

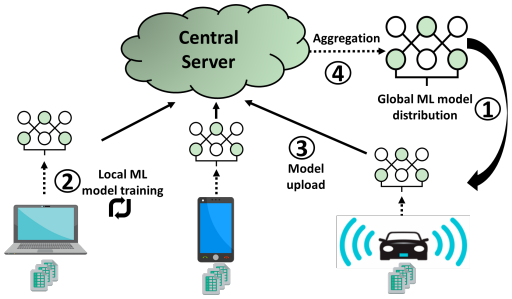


Figure 1.4: Federated Learning (FL)

aggregates these updates to form an improved global model. This process repeats until the model converges.

FL offers users greater control over their data and facilitates privacy by design through distributed training and aggregation across a network of client devices. As networks evolve towards a unified compute continuum (see Sec. 1.1), this thesis investigates the convergence of these advancements, comparing emerging DML schemes with traditional CL pipelines. It also explores trade-offs and practical considerations for real-world implementations, with a particular focus on applications in the automotive domain.

2

Centralized *vs.* Federated Learning comparison

2.1 Introduction

ML workloads in networked environments have historically been dominated by centralized (CL) approaches. In these models, clients gather raw data—such as measurements, audio, images, or video frames—from their environment, preprocess it, and transmit it to a centralized server. The server then performs computationally intensive ML tasks, such as model training. However, advancements in networking technologies, such as Edge Computing [21], coupled with the growing emphasis on data privacy through regulations like the General Data Protection Regulation (GDPR) [22], have spurred the development of distributed (DML) approaches as viable alternatives to CL for supporting ML in networked settings.

Both CL and DML impose significant demands on network resources. Training data volumes range from gigabytes to terabytes [23], and the computational demands can reach thousands of teraFLOPS [23]. This raises a critical question: which ML paradigm is better suited for accurate model training while adhering to network resource constraints? Furthermore, what are the advantages and disadvantages of adopting a DML scheme compared to traditional CL?

The choice between ML paradigms is a complex, multi-faceted problem influenced by numerous factors. A network ecosystem consists of diverse stakeholders, such as clients, network operators, and cloud or edge providers, each with unique constraints and priorities. These include client-specific considerations (*e.g.,* data distribution properties, device mobility, and battery life), network-related factors (*e.g.,* bandwidth, latency, and packet loss), and cloud-related limitations (*e.g.,* energy consumption and processing capacity). Additionally, the inherent complexities of ML workflows—such as hyperparameter tuning, client selection strategies, and data preprocessing—further complicate the decision-making process.

To the best of our knowledge an in-depth analysis on the ML scheme selection problem is missing from the current literature. Prior works have attempted to address similar questions in a restrictive manner. There is an extensive theoretical [24], [25] and experimental [26], [27] research on the comparison between CL and DML in terms of convergence, but the relevance of the two approaches to the underlying network is widely neglected. As such, those efforts lack an *end-to-end* system-parameter analysis to shed light on the involved (network) resources trade-offs (affecting various stakeholders).

Table 2.1: Research directions in ML schemes comparison

ML schemes comparison			
CL vs FL		**CL/FL vs other DML schemes**	
Investigation parameter	*Research works*	*Investigation parameter*	*Research works*
convergence rate	[24], [27]	convergence rate	[28]
accuracy	[27], [26], [29]	accuracy	[18], [30], [31], [19]
training loss	[24]	energy cost	[18], [30]
ML hyperparameters	[25]	bandwidth cost	[30]
data distribution	[25]	privacy & security	[30]

This chapter addresses the ML scheme selection challenge from a system-centric perspective, offering a holistic network resource analysis that accounts for all stakeholders. Given the diversity of ML paradigms, we narrow our focus to two extremes: the well-established CL approach and the emerging, widely adopted Federated Learning (FL) framework [20]. In FL, clients collaboratively train models while retaining their data locally, with a central server orchestrating the process, thereby enhancing data privacy. For this chapter, we relax the data privacy constraint to enable a fair comparison between CL and FL, emphasizing their respective impacts on model accuracy and network resource consumption.

Our investigation centers on the following research questions: 1) How do ML hyperparameter choices influence resource consumption in each scheme?, 2) Given the ratio of client data volume to model size, how do accuracy, convergence speed, and stakeholder resource consumption vary between the schemes?, 3) How do the schemes compare in terms of convergence speed over time?, 4) How do they scale with an increasing number of participating clients, particularly in terms of accuracy and convergence speed? and 5) How does heterogeneity in client data affect each scheme's accuracy?

To address these questions, we developed a cloud-to-client system model based on empirical measurements and commercial benchmarks, replicating real-world performance in terms of bandwidth availability, energy consumption, and processing capacity. Two network scenarios are considered: a mobile LTE network and a Wireless Local Area Network (WLAN), with realistic mobility patterns derived from real-world traces. Additionally, a dedicated AI/ML software environment is employed to simulate the training processes of CL and FL.

The remainder of this chapter is structured as follows: Section 2.2 reviews the relevant literature. Section 2.3 presents our system model. In Section 2.4, we discuss the results of our simulations. Finally, Section 2.5 concludes the chapter.

2.2 Related work

The debate surrounding the adoption of centralized versus distributed ML schemes in networked environments remains under-explored. Among distributed ML approaches, FL has garnered significant attention as a compelling alternative to CL due to its robust data privacy guarantees (see Table 2.1). However, existing studies primarily focus on comparing CL and FL in terms of model performance, such as accuracy or convergence, without adequately addressing their impact on network resources.

For instance, in [26], FL is applied to intrusion detection in an IoT environment. The decentralized scheme achieves comparable accuracy to CL (with a maximum difference of 5%) while preserving data privacy. Similarly, FL's applicability to sensitive medical data

is explored in [29], using three benchmark clinical datasets. The results indicate that FL approaches CL's performance across metrics like accuracy and recall, with a maximum discrepancy of 4

In addition to accuracy, FL and CL are often evaluated based on convergence rates. In [24], Distributed Stochastic Gradient Descent (D-SGD) achieves comparable training loss to CL in a static network topology while demonstrating faster convergence in low-bandwidth environments. In another study [27], a network of distributed workers, with identical hardware capabilities, is evaluated on the Fashion-MNIST dataset for image classification. CL achieves 2% higher accuracy than FL, while FL converges nearly 40% faster. However, the assumption of identical hardware across workers limits the applicability of these findings to real-world scenarios involving heterogeneous, resource-constrained devices. To address this, [25] benchmarks various FL algorithms *e.g.,* FedAvg, Cooperative, *etc.*, against CL, showing that CL consistently delivers more accurate classifications. However, FedAvg demonstrates potential, provided the client data is independent and identically distributed (i.i.d.).

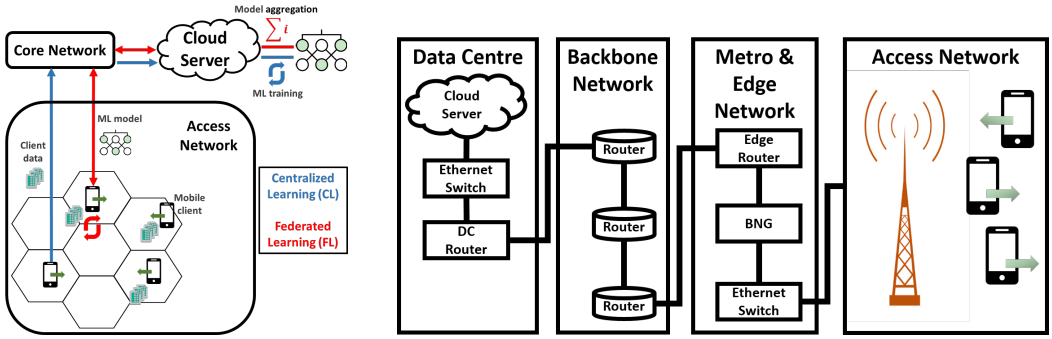
Beyond FL, other distributed ML schemes have been proposed as alternatives to CL, including hybrid and collaborative approaches (see Table 2.1). Hybrid schemes, which blend characteristics of CL and FL, are seen as a middle ground. For instance, [18] introduces a hybrid scheme that achieves accuracy levels comparable to CL while surpassing FL by an average of 10%. Additionally, this scheme balances the energy efficiency of FL with the higher energy demands of CL. In another study [30], a hybrid edge-FL scheme employs blockchain mechanisms to enhance security. This approach outperforms both CL and FL in terms of accuracy (with a maximum improvement of 5%) while reducing energy consumption by nearly 50% compared to CL, albeit with a 25% increase in bandwidth usage.

Collaborative ML approaches represent another research direction aimed at reducing communication costs by eliminating the need for a central server. Examples include Swarm Learning [31], where clients (swarm nodes) share model parameters directly via a peer network. Tests on clinical data demonstrate improved accuracy over CL for various patient detection scenarios, such as leukemia and COVID-19. Similarly, Gossip Learning [28] enables clients to exchange small portions of their model parameters with neighbors, achieving comparable accuracy to FL, albeit with slower convergence. Finally, Peer-to-Peer Learning [19] facilitates direct model exchanges among clients using a secure sharing protocol. While this method's convergence heavily relies on i.i.d. data, its accuracy is comparable to CL.

While hybrid and collaborative schemes show promise in improving accuracy over FL, they often neglect the impact on system resources. This oversight is problematic, as efficient ML deployment in networked environments requires a balance between accuracy and resource consumption. Without addressing the network resource implications, the practical feasibility of these ML schemes remains uncertain. Our work seeks to bridge this gap by investigating the resource consumption of ML schemes, specifically comparing CL and FL. Through an end-to-end analysis, we aim to illuminate the trade-offs between accuracy, convergence, and resource efficiency, providing a more comprehensive understanding of the performance and practicality of ML schemes in real-world network environments.

2.3 System model overview

Our system model consists of multiple mobile clients operating within a dynamic network environment. Each client possesses sensor-captured data that serves as input for the given



(a) (b)

Figure 2.1: (a) Centralized (CL) and Federated Learning (CL) architecture (b) Network architecture

ML task. These clients are connected to a central entity, such as a cloud server located in a DC, via an intermediate core network, enabling data exchange with the server. To simplify the analysis, we exclude control and management plane messages (*e.g.,* coordination and signaling), as their size is negligible compared to the actual training data. This exclusion does not compromise the model's accuracy. The system's primary objective is to execute an ML task over multiple communication rounds, leveraging the data available on the clients through the following ML schemes:

Centralized Learning (CL)

In each training round the server selects a group of (mobile) clients (see the green horizontal arrows in Fig. 2.1a) which upload their data directly to the server (see the blue vertical arrows in Fig. 2.1a), as described in Sec. 1.2. The training task is thereafter performed centrally by the server using all client data acquired in that round. This process repeats for several rounds, each time with a selection of a random group of clients, until a predefined time limit is reached or all data is depleted.

Federated Learning (FL)

The server broadcasts the training model to the selected clients (learners), which in turn are responsible to train it using their own (private) data and computing resources (see the red recursive arrows in Fig. 2.1a), as described in Sec. 1.2. As opposed to CL, clients are no longer required to upload their actual data to the server. This is a key restriction posed by FL, in order to preserve user-data privacy. Instead, once the training is completed, they upload the updated model parameters to the central server, which in general are considerably lightweight compared to the actual data. Upon collecting the updated model parameters, the server performs model aggregation (see red vertical arrows in Fig. 2.1a) and re-distributes the updated aggregated model to another group of clients.

Standard Learning (SL)

While the ML task is performed in a centralized location, CL is not to be confused with the standard ML paradigm, where training is performed using the entire dataset. The latter is referred to as Standard Machine Learning (SL). Essentially, an implementation of SL would require for the server to wait for all participating client datasets to arrive in

order to perform the training task. On the other hand, periodic training *i.e.*, on a per round basis occurs in CL, given that (recently generated) data arrives in every round. In our study we focus on CL, rather than SL, since the way SL functions deems it impractical for employment in a real system. Applications either cannot afford waiting for the entire dataset to reach the DC or are agnostic to the entire client dataset. More importantly SL lacks adaptability, since it does not produce a global ML model on a per-round basis.

In an actual system, clients do not generate (or cannot acquire) a single dataset, which will be uploaded once; they rather repeatedly collect data *e.g.*, user analytics, network measurements, *etc.*, which can be (potentially) streamed to the cloud in a continuous manner. Moreover, depending on the environment, this data can change in time *e.g.*, traffic scene images and therefore the output of the ML task *i.e.*, the ML model, needs to be updated on a regular basis. As such, SL is not proposed as a practical solution, but serves as a baseline to the evaluation of other centralized schemes. Moreover, CL is needed in our study to ensure a fair and meaningful comparison to FL, given that a global ML model is generated in every round.

In the following Sections (2.3.1-2.3.6) we detail the elements that comprise our system model. All involved quantities are summarized in Table 2.2.

Table 2.2: Nomenclature of involved quantities

Symbol	Definition
d_{sample}	Training dataset samples
d	Training dataset size (bytes)
z	Training dataset partitions (*i.e.*, total clients)
r_{sample}	Training dataset size to samples ratio
m	ML model size (bytes)
r_{data}	Client data to model size ratio
k	Per round participating clients
q	Online clients
t_{upload}	Total time for client data upload in each round (sec)
t_{end}	Maximum duration of ML task (sec)
s_{client}	Client throughput - access network (bytes/sec)
c^{CH}	Average area throughput - access network (bytes/sec)
σ	Client throughput standard deviation - access network
c^{CH}_{min}	Minimum client throughput - access network (bytes/sec)
s_{core}	Core network element's throughput (bytes/sec)
σ_{ed}	Data skewness parameter

Symbol	Definition
σ_{iid}	Independently and identically distributed (i.i.d) level shape parameter
v^{ML}_{client}	Client device computational capacity for training (samples/sec)
v^{ML}_{cloud}	Cloud computational capacity for training (samples/sec)
v^{AG}_{cloud}	Cloud computational capacity for model aggregation (models/sec)
e_{client}	Total energy expenditure for all clients (J)
$p^{TX}_{client_i}$	Client device power consumption for transmission (W)
$p^{RX}_{client_i}$	Client device power consumption for reception (W)
$p^{ML}_{client_i}$	Client device power consumption for training (W)
e_{core}	Total energy expenditure in the core network (J)
e_{cloud}	Total energy expenditure in the cloud (J)
p^{ML}_{cloud}	Cloud power consumption for training (W)
p^{AG}_{cloud}	Cloud power consumption for model aggregation (W)
h_{epochs}	Number of epochs (ML hyperparameter)
h_{batch}	Batch size (ML hyperparameter)
h_{rate}	Learning rate (ML hyperparameter)

2.3.1 Network architecture and attributes

The network includes the wireless (radio) and the wired part (see Fig. 2.1b). Two common cases are considered for the wireless part; a mobile LTE and a WLAN network. In the LTE case, a typical cellular architecture is assumed, where a basestation (BS) lies in the centre of each cell. In the WLAN case, several access points (APs) are assumed, which cover a local network area (coverage area). We refer to the wireless part (link between clients and BS/AP) as the access network. The wired part (from BS/AP up to the cloud)

comprises the core network (see Fig. 2.1b). The core network includes: 1) the metro and edge network, where edge nodes reside, 2) the backbone network, which includes core switches and routers and 3) the infrastructure to reach the DC's cloud server.

Access network throughput

The client throughput s_{client} in both the uplink (UL) and the downlink (DL) direction (in MBytes/sec) is modelled as a Gaussian random variable ($\tilde{\text{N}}$). Its mean value is equal to the average area throughput c^{CH} (where CH marks the corresponding radio technology *i.e.*, LTE or WLAN), divided by the number of online clients q. The inclusion of the online clients in the computation allows to factor-in the way the locally-present number of users shapes the throughput provision in the considered area. c^{LTE} refers to the average cell throughput; that is 5.9 (UL)/7.73 (DL) MBytes/sec for 2.5 GHz LTE according to [32]. c^{WLAN} on the other hand refers to the average coverage area throughput, which is 2.25 (UL)/2.38 (DL) MBytes/sec for IEEE 802.11g, according to [33]. The standard deviation parameter σ is set to 20% of the client's mean throughput based on [34] and accounts for throughput variations *e.g.*, due to path loss and interference. Given that a client is found at a certain moment connected (online), we also assume that there exists a minimum throughput threshold (c^{CH}_{min}) to enable server-client communication, both in DL and in UL. c^{LTE}_{min} is assumed equal to the 5% cell edge rate; that is, 0.24 (UL)/0.22 (DL) MBytes/sec according to [32], which represents the worst-case scenario for the respective radio conditions. For WLAN, a similar value for the coverage area (c^{WLAN}_{min}) can be obtained; that is, 0.03 (UL-DL) MBytes/sec according to [35]. Thus, s_{client} for UL and DL, is given by:

$$s_{client} = max\{\tilde{N}(\frac{c^{CH}}{q}, \sigma), c^{CH}_{min}\} \tag{2.1}$$

Core network throughput

The core network includes the following elements [36]: 1) An interface to the access network (BS for LTE or AP for WLAN); 2) The metro and edge network's elements *i.e.*, an ethernet switch, a broadband network gateway (BNG) and the edge router; 3) the backbone network's routers and 4) the DC's elements *i.e.*, an edge router and a data center switch. The average throughput of each element for the UL/DL (s_{core}) is based on Cisco routers/switches performance benchmarking [36] and measurements on access network interfaces (3-sector 2×2 Multiple-Input-Multiple-Output remote radio 4G/LTE) [36]. A total number of 3 backbone network's routers is considered, as a hopcount of maximum 3 in the backbone network suffices to reach the DC for the majority of popular services *e.g.*, Facebook, Bing, Google, *etc.* [37].

2.3.2 Client mobility pattern

In contrast to synthetic or theoretical mobility models, real-world traces offer more realistic performance evaluations and reliable results. However, questions often arise regarding their statistical representativeness and the generalizability of the findings. To address this, conducting multiple iterations using publicly available datasets enhances both the credibility and the broader applicability of our results. For our system model, we have chosen the Shanghai Telecom Dataset [38] for LTE traces and the Wifidog [39] for WLAN traces. The Shanghai Telecom Dataset contains records of UEs accessing the Internet through

a BS in a period of 15 days. The database records timestamps for connection initiation and termination at one-minute intervals, corresponding to the dataset's time granularity. This allows for the calculation of clients' online presence. Clients are monitored and considered online as long as they remain connected to the network. When a client moves to another cell *i.e.*, serviced by another BS, a handover (HO) is assumed, to capture service continuity. We ignore any communication disruption during the HO process. However, a change of cell will affect the client throughput, which depends on the corresponding cell's congestion (see Sec. 2.3.1). The Wifidog dataset comprises user session traces collected from various free WiFi hotspots in Montreal, Quebec, Canada. Similar to the Shanghai dataset, each node represents an access point (AP), and timestamps are recorded for user login and logout events with a granularity of one second. Clients are considered online during the period they remain connected to a hotspot.

2.3.3 Client data acquisition and distribution

Client devices acquire raw data via their sensors, cameras *etc.*, which can be used for training. The acquisition is shaped by the client's acquisition rate, the UE's storage capacity and the data staleness level [40]. To emulate such a behavior, we divide our image-classification training dataset (see Sec. 2.3.6) into z partitions, which are assigned to the selected clients, representing the data that the clients have generated and stored on their devices. We define the dataset size to samples ratio $r_{sample} = d/d_{sample}$, where d_{sample} is the total number of dataset's training samples and d the dataset's size in Bytes. Marking the per client dataset size as d_i, the total dataset can be written as: $d = \sum_{i=1}^{z} d_i$. Assuming a fixed ML model size m, the per client data to model size ratio is defined as $r_{data_i} = d_i/m, i \in [1, z]$. r_{data} is a key parameter in the ML scheme selection problem. Not only it regulates the network's resource consumption *e.g.*, large chunks of data require more bandwidth in order to be uploaded (CL) or more energy in order to be processed (FL), but can also affect the convergence of the ML task itself. Besides data acquisition, we also explore how this data is distributed across clients *i.e.*, data heterogeneity. We focus on two dimensions; variations in size are modeled by the evenly distributed level (e.d.), while variations in content are captured by the independent and identically distributed level (i.i.d.). Generally, client data can experience various levels of e.d., i.i.d. or combinations of both.

On the e.d. level

The e.d. level describes the dataset size distribution across clients. The size of each client's dataset is modeled as a random variable ($\tilde{F}$) that follows Zipf's law, in line with related research [41]. To represent a random variable following Zipf's law, we are using Zeta distribution, which has a probability density function of $p(x) = \frac{x^{-(\sigma_{ed})}}{\zeta(\sigma_{ed})}, \sigma_{ed} \in (1, +\infty)$. ζ represents the Riemann Zeta function, while the Zeta distribution's skewed parameter $\sigma_{ed} \in (1, +\infty)$ shapes the e.d. level, moving from a uniform data distribution ($\sigma_{ed} >> 0$) towards higher asymmetry cases (σ_{ed} close to 1); an additional restriction is also imposed that the minimum dataset size equals the size of one batch h_{batch} (see Sec. 2.3.6), in order to ensure that training can occur at all cases. Therefore the client dataset size becomes $d_i = max\{h_{batch}, \tilde{F}(\sigma_{ed})\}, \sigma_{ed} \in (1, +\infty), \forall i \in [1, z]$. If the initial dataset is evenly distributed among the clients ($\sigma_{ed} >> 1$), the client dataset size d_i and dataset to ML model size ratio r_{data_i} are simplified to: $d_i{=}d/z$, $r_{data_i}{=}d/(m \cdot z), \forall i \in [1, z]$, respectively. A σ_{ed} value close to 1 on the other hand marks a setting where few clients hold considerable amounts of data.

On the i.i.d. level

If a setting with independent and identically distributed (i.i.d) data is assumed, then the data samples in each client have the same probability distribution and are mutually independent. In our image-classification problem (see Sec. 2.3.6) such a setting would essentially mean that each user holds samples from all classes (unbiased setting). That is represented by the i.i.d. level σ_{iid}, being equal to the total number of dataset classes. For some real-world scenarios, non-i.i.d. (biased) settings could occur, since each participating client might not be expected to possess a representative subset of all classes in the total training dataset. To study different levels of bias, we restrict the number of dataset classes a client can hold *i.e.*, the value of σ_{iid} is smaller compared to the total number of classes.

2.3.4 Device computational capacity

User equipment

The computational capacity of a mobile device to perform an ML task v^{ML}_{client}, measured in (processed) training samples/sec depends on the dataset content *e.g.*, images pose different requirements than natural language, the UE capabilities and the complexity of the ML model. We use a reference (average) value of v^{ML}_{client}=125 training samples/sec, as the most appropriate for our training dataset, ML model and ML hyperparameter settings (see Sec. 2.3.6), based on approximations for popular large-scale classification tasks [42].

Cloud server

Computational tasks for the cloud server include training (in CL) and ML model parameter aggregation (in FL). Regarding training, we assume that a DC is equipped with a Tensor Processing Unit (TPU), which demonstrates an average computational capacity for training $v^{ML}_{cloud} = 40K$ training samples/sec [43]. In regards to aggregation, no reference values could be found in the literature, thus we rely on an empirical approach; we measure the average capacity for training and aggregation tasks in our setup *i.e.*, 6250 training samples/sec and 1.56 ML model aggregations/sec respectively and compare against the training capacity reference value of 40K training samples/sec [43]. Assuming a linear relation, the average cloud aggregation capacity becomes v^{AG}_{cloud}=10 model aggregations/sec. Our assumptions do not cover scale-out schemes, where clusters of servers may be used to increase parallelism in DCs.

2.3.5 Device energy consumption

User equipment

Any consumption related to the UE's standard operation *e.g.*, the device's operating system functionalities or displaying, is neglected and we focus on energy expenditure due to transmission (TX)/ reception (RX) of data and ML processing (training tasks). The energy consumption e_{client_i} *i.e.*, battery discharge of the i^{th} client's device is computed as: e_{client_i}=$e^{TX}_{client_i}$+$e^{RX}_{client_i}$+$e^{ML}_{client_i}$, where the superscript TX, RX and ML marks one of the aforementioned functions. In a given time period t, this can be calculated as $e_{client_i} = p_{client_i} \cdot t$, where $p_{client_i}, i \in [1, z]$ stands for the respective (average) power consumption. Average power consumption values related to transmission are reported in [44], where $p^{TX}_{client_i}$=2.2 Watts and $p^{RX}_{client_i}$=1.5 Watts, $\forall i \in [1, z]$ for LTE and $p^{TX}_{client_i}$=0.75 Watts and $p^{RX}_{client_i}$=0.25 Watts for WLAN. Likewise, for ML, based on [42], we assume

$p^{ML}_{client_i}$=2 Watts, $\forall i \in [1, z]$, as the most appropriate to our ML model's hyperparameters and our choice of training task, being an image classification problem (see Sec. 2.3.6). The sum of all device energies e_{client_i} comprises the total client energy expenditure e_{client}.

Core network devices

Energy consumption in the core network e_{core} is calculated by summing the energy consumption per core component (see Fig. 2.1b) *i.e.*, all routers, gateways and switches of the backbone, metro & edge network and the cloud's plus access network's interfaces (BS for LTE and AP for WLAN), which in turn is given by [36] in relation to the data exchanged in the UL/DL direction (as average values measured in Joules/bit).

Cloud server

The computational capacity values of the main tasks that run in the cloud *i.e.*, ML training (in CL case) and model aggregation (the latter occurs in the FL case) are discussed in Sec. 2.3.4. Energy expenditure per task in the cloud server can thus be calculated, given an average power expenditure. Note that inference *i.e.*, applying the trained ML model on new data also involves resource utilization, but may come as a stand-alone task, much later than training which is far more demanding in computations and network resources. Similarly, latency is not considered in our setup as training time requirements typically dominate over any latency considerations. For the training task (in CL), being an intensive processing task, we assume an average power p^{ML}_{cloud}=384 Watts, based on Google's TPU benchmarking [45]. For the (less-intensive) aggregation task (FL), we assume p^{AG}_{cloud}=15 Watts, based on measurements for matrix multiplication tasks [46], which are similar in complexity to weighted averaging (aggregation). The cloud energy consumption e_{cloud} then becomes:

$$e_{\text{cloud}} = \begin{cases} p^{ML}_{cloud} \cdot d/(v^{ML}_{cloud} \cdot r_{sample}), & \text{for CL} \\ p^{AG}_{cloud} \cdot z/(v^{AG}_{cloud}), & \text{for FL} \end{cases} \tag{2.2}$$

2.3.6 Machine Learning task

The ML schemes are evaluated using a representative ML task for vehicular applications: image classification. This task is extensively utilized for various perception functions in autonomous and remote driving systems [47]. Specifically, our evaluation is based on the Street View House Numbers (SVHN) dataset, a benchmark in prior studies *e.g.*, [48], [49]. The SVHN dataset consists of real-world images of digits extracted from natural scenes (house numbers captured in Google Street View). It includes 531,131 32x32 color training images (1.3 GB in size) divided into 10 classes (digits 0–9) and 26,032 test images (63 MB in size). The division between training and test data is predefined by the dataset creators.

For the image classification task on SVHN, we designed a custom artificial neural network applicable to both the CL and FL scenarios. In the FL case, we employed the Federated Averaging (FedAvg [20]) algorithm for model aggregation, as implemented by the PySyft framework [50]. In PySyft, a uniform averaging method is used *i.e.,* all ML models are equally weighted during aggregation—unlike the original FedAvg algorithm, which applies weighted averaging based on the number of training samples per client.

Our neural network consists of three layers: an input layer with 3,072 neurons corresponding to the total pixels in the SVHN images (32x32x3), a hidden layer with 512 neurons using Rectified Linear Unit (ReLU) activation to filter non-positive values, and

an output layer with 10 neurons (one for each SVHN class) employing LogSoftmax activation [51] for efficient multi-class classification. To calculate the training loss, we selected the negative log-likelihood loss function [51], which pairs effectively with LogSoftmax in classification tasks. The total size of the ML model is approximately 6.1 MB.

Although more complex and larger models, such as convolutional neural networks (CNNs) like MobileNet, ResNet, or DenseNet [51], could be used to achieve higher accuracy for image classification tasks, they are less practical for resource-constrained user equipment (UE). Constraints such as limited disk space, processing power, memory, and battery capacity could deter user participation in the training process, reducing the realism of the system. Therefore, our design prioritizes practical feasibility over model complexity.

Hyperparameter tuning

Setting the hyperparameters of an ML model—such as the number of epochs, batch size, and learning rate—is a crucial preparatory step before commencing the ML training process. These hyperparameters define the training methodology rather than being intrinsic components of the resulting ML model. Instead, they influence how the model's parameters (referred to as the resulting ML model) are adjusted during training. Optimizing these hyperparameters involves conducting multiple test runs with varying combinations of values. The configuration that delivers the highest accuracy is then applied to the main training task. Neglecting this tuning phase can lead to significant performance degradation or even failure to achieve convergence, as demonstrated in [52]. While most studies focus on optimizing hyperparameters based solely on the resulting ML model's performance metrics (*e.g.,* accuracy or convergence speed), our work also considers the impact of hyperparameter selection on network energy consumption, particularly as influenced by processing demands.

To this end, our analysis begins with the total number of epochs (h_{epochs}), which determines how many times the algorithm processes the entire training dataset. This parameter influences not only the final performance of the ML model but also the processing time, and hence, the energy consumed—whether by the clients in FL or by the cloud server in CL. Once the value of h_{epochs} is established, we further examine two other critical hyper-parameters: the batch size (h_{batch}) and the learning rate (h_{rate}). The batch size defines the number of training samples used in a single iteration (forward and backward pass), while the learning rate parameter controls the magnitude of weight updates during training, effectively determining the step size of the model's parameter adjustments. Both hyper-parameters primarily influence the ML model's accuracy, while also having a secondary impact on processing time and energy consumption. By addressing these parameters comprehensively, our study aims to evaluate their dual effect on model performance and resource efficiency.

2.3.7 Emulation environment process

The following steps describe how the above-mentioned modelling components, together with client selection and server-client communication are incorporated in our emulation environment, both for the CL and the FL case. Initially, the cloud server generates a (non pre-trained) ML model, which seeks to train by utilizing available client data. The central server orchestrates, synchronizes and controls the training process and the clients at all times. Also, the SVHN dataset (see Sec. 2.3.6) is split into z partitions (see Sec. 2.3.3). We then run a series of communication rounds, until all partitions are used or a

predefined time deadline (t_{end}) is reached. The time deadline is selected, based on the respective time granularity of each mobility dataset. In each communication round:

Step 1: The server identifies all (available) online clients (q) determined by the corresponding mobility dataset that captures the mobility dynamics.

Step 2: A total of $k < z$ clients are randomly selected, as a subset of q, each of which is assigned a partition (see Sec. 2.3.3). If there are not enough online clients present to satisfy our selected per round participating client number ($q < k$), all the currently online clients are used. If no clients are found at all ($q = 0$), the round is terminated and a waiting period is introduced, equal to the mobility dataset's time granularity; that is 60 sec for the Shanghai Telecom and 1 sec for the Wifidog dataset (see Sec. 2.3.2).

Step 3: The ML scheme is initiated: In case of CL, the selected clients upload their (raw) local datasets to the cloud server. The time (t_i) required for each client's dataset ($d_i, i \in [1, z]$) upload is equal to the time of the access network upload plus the time of the core network upload, therefore can be written as (see Sec. 2.3.1): $t_i = d_i/s^{UL}_{client} + \sum_j (d_i/s^{UL}_{core_j})$, for j core components. A parallel communication protocol is assumed, thus the total time to upload all datasets (t_{upload}) equals to that of the "slowest" client: $t_{upload} = max\{t_i\}$. Upon collection, the server merges all round's data into a super-dataset, which is then randomly shuffled to mitigate the effects of variance [53]. With the collected super-dataset the server trains the ML model. Training time is calculated using the cloud server's computational capacity model (see Sec. 2.3.4). The completion of the ML training marks the end of the round.

For the FL case, the cloud server firstly shares the training models (of size m) to the selected clients (k). Afterwards, each client trains the model, utilizing its assigned dataset and uploads the updated model back to the server. When all models are uploaded back to the cloud server (again in a parallel manner, similar to the CL case), the server performs model aggregation (averaging). The time needed for the federated training and the server's aggregation task is calculated from the UE's and cloud servers computational capacity model respectively (see Sec. 2.3.4).

Step 4: A communication failure occurs when a client goes offline, while performing a task (either during training or during data exchange), irrespective of the task's completion percentage. Such a failure is checked in our emulation environment by parsing the corresponding mobility dataset. In case of failure, the client's contribution is neglected by the cloud server (training for CL/aggregation for FL). However, to account for the time/resources spent for the partial communication, we consider a delay time equal to the estimated task's time, along with the respective resource consumption, assuming the worst-case scenario *i.e.* that the connection is lost, when the task was almost completed.

2.4 Simulation results

2.4.1 Simulation setup and evaluation metrics

The simulation environment was set up on a single desktop machine with the following specifications: Intel Core i7-10700 processor (2.9 GHz), 64-bit architecture, 16 GB of Random Access Memory (RAM), and Windows 10 as the operating system. To simulate the distributed learning environment, we used the PySyft library [50]. PySyft is an open-source, Python-based framework designed for secure and private machine learning. It allows for the decoupling of private data from the model training process by employing techniques such as Federated Learning, Differential Privacy, and Encrypted Computation. PySyft's interface is similar to that of the numpy Python library [54] and integrates

seamlessly with popular deep learning frameworks.

The DML environment[1] is wrapped using a custom Python-based discrete event simulator, which emulates the underlying network *i.e.*, the mobile clients, the core network and the cloud server, as illustrated in Fig. 2.1b. The network simulator software realizes the network throughput, client mobility, data distribution, computational capacity and energy consumption models, as presented in Sec. 2.3. Its clock is synced to the mobility dataset's timeframe, which is used as global (time) reference. A deadline of $t_{end} = 24hrs$ is chosen across all experiments which are set to terminate when all dataset (SVHN) partitions are used or t_{end} is reached. We report that in 95% of the cases, termination occurred due to dataset depletion, while the average termination (simulation) time was 4 hrs, throughout all our experiments. Unlike the majority of works in literature that evaluate the performance of CL *vs.* FL with regards to their accuracy, we employ a broad set of carefully-selected metrics to capture all previously-overlooked dimensions.

Test Accuracy

The effectiveness of the trained ML model is evaluated on the SVHN test data (see Sec. 2.3.6). The percentage of successful to total classifications provides the test accuracy metric, with a maximum value of 100%. Both in CL and in FL, we assume that the cloud server has the ability (and the capacity, given the ML model's negligible size) to save each round's ML model, so that the most accurate ML model can be extracted in the end of the experiment. Keeping track of each round's output is a standard technique in ML training [55], to avoid over-training and therefore parameter over-fitting, which results in less accurate ML models. Thus, the term Test Accuracy refers to the maximum accuracy achieved during the (CL/FL) experiment and not necessarily the accuracy of the chronologically latest ML model.

Traffic Overhead

It is defined as the total amount of data exchanged between the cloud server and the clients for the duration of the ML process, multiplied by the total number of hops the data traverses, after leaving its origin node *i.e.*, $hops = (total\ network\ nodes) - 1$. In case of CL, the exchanged data refers to the raw data uploaded by the clients, while in FL it refers to the ML model's parameters uploaded by the clients, plus the ones distributed back to clients by the cloud server (see Fig. 2.1a). For the sake of clarity, traffic overhead is normalized to the total dataset size *i.e.*, 1.3 GB. Traffic overhead reflects the total bandwidth consumed during the ML process, accounting for both successful and unsuccessful communications. On the contrary, the total overhead generated by communication failures (see Sec. 2.3.7), is defined as *Traffic Loss*.

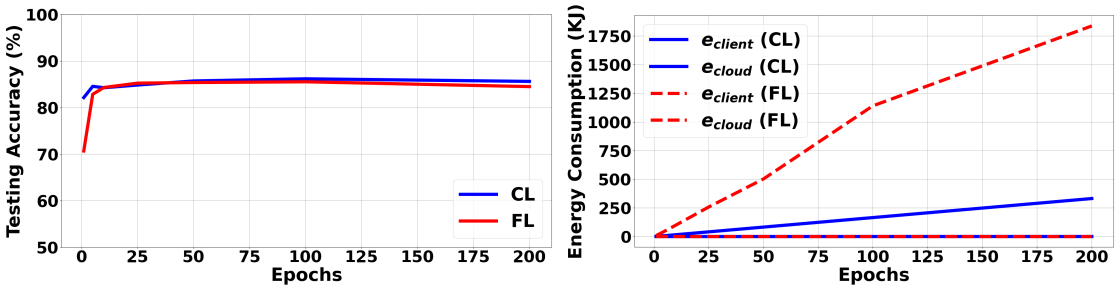
Energy Consumption

It captures the energy expenditure in all involved devices, during the ML process (Sec. 2.3.5), regardless the success/failure of transmission. Specifically, for the clients, we consider the total energy of all devices involved (due to ML processing and transmission/reception). For the core network it is only limited to expenditure due to data exchange. Lastly, for the cloud server, we only account for consumption due to processing (either model aggregation in FL or ML training in CL). Similarly to the definition of *Traffic Loss*, we also define the *Energy Loss*, as the total energy consumed *e.g.*, by a client's

[1]The source code is publicly available at: https://github.com/giorgosdrainakis/dml

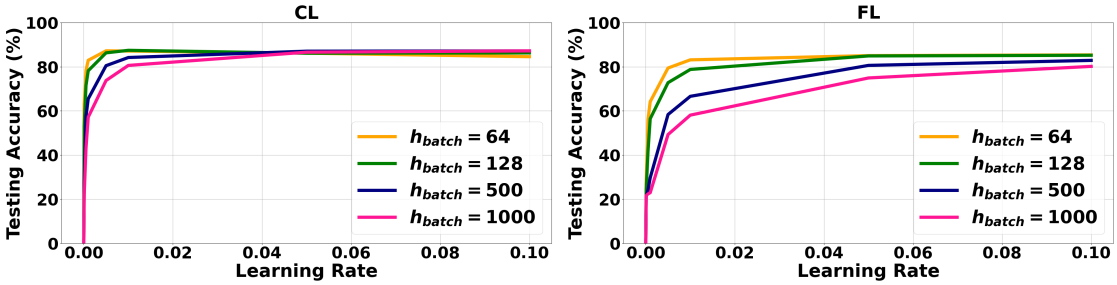
device for training a model in FL, but the result was not utilized due to a subsequent client-server communication failure.

2.4.2 CL *vs.* FL: Energy-aware hyperparameter exploration



(a) Testing accuracy w.r.t. epochs (b) Energy expenditure w.r.t. epochs

Figure 2.2: ML hyperparameter tuning: Number of epochs



(a) Testing accuracy w.r.t. batch size and learning rate (CL) (b) Testing accuracy w.r.t. batch size and learning rate (FL)

Figure 2.3: ML hyperparameter tuning: Batch size and Learning Rate

The exploration of the ML hyperparameter values (tuning) is performed prior to the actual ML tasks (experiments), as discussed in Sec. 2.3.6. Starting from the number of epochs h_{epochs} we consider the following values: {1, 5, 10, 25, 50, 100, 200}, which are commonly used in bibliography for image-classification tasks and SVHN specifically [49]. Our test-runs suggest that our ML model achieves a maximum accuracy of 85%, if $h_{epochs} \in [25, 100]$ both for CL and FL (see Fig. 2.2a). When looking to the underlying energy consumption in the clients side, one observes that an increase on h_{epochs} causes a linear increase in energy consumption, with angles of incline equal to 0.46 and 0.78 rad for CL and FL, respectively (see Fig. 2.2b). To account for both accuracy and the consumed energy, a value of h_{epochs}=25 is selected.

Moving to the batch size h_{batch} and learning rate h_{rate}, we perform an exhaustive search over the following values: {64, 128, 500, 1000} for h_{batch} and {0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1} for h_{rate}, based on [48] and [49]. In the CL case (see Fig. 2.3a), the maximum accuracy is achieved for a combination of $h_{batch} \geq 128$, $h_{rate} \geq 0.05$ and for the FL case (see Fig. 2.3b) $h_{batch} \leq 128$, $h_{rate} \geq 0.05$. For a fair comparison of the two ML schemes (CL, FL), we assume a common set of values of $h_{batch} = 128, h_{rate} = 0.1$. The selected hyperparameter values are kept fixed throughout our experiments.

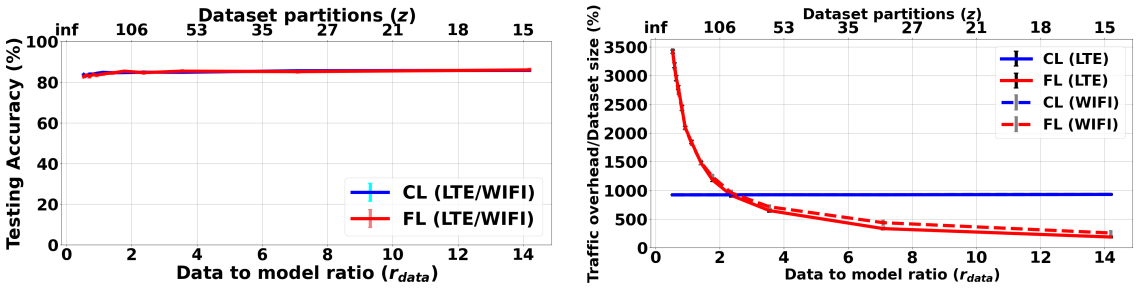
2.4.3 CL *vs.* FL: The effect of client data to model size ratio r_{data}

In this section, we investigate the way the per client data to model ratio r_{data} shapes the performance of the ML schemes under evaluation. Essentially, r_{data} indicates the amount of data a client holds (in relation to the fixed ML model size m). For this set of experiments, we assume data is symmetric across all clients (*i.e.*, i.i.d. and e.d. setting) and vary the number of dataset partitions *i.e.*, total clients z ={15, 30, 60, 90, 120, 150, 190, 230, 260, 300, 330, 360, 400}. Therefore the data to model ratio becomes r_{data_i}=$d/(m \cdot z), \forall i \in [1, z]$ (see Sec. 2.3.3). We also, for now, assume a constant value for the per round participants k=5. Keeping in mind that the total dataset (SVHN) size d is fixed to 1.3 GB (see Sec. 2.3.6), larger values of r_{data} represent setups, where fewer clients with larger dataset partitions (z) participate in total, in the ML scheme. Vice versa, smaller r_{data} values represent a situation, where more clients with fewer data participate in total. Nevertheless, the aggregated (total) training data (SVHN) in each experiment remains the same, ensuring a fair comparison for all training tasks. Each of the 13 experiments (representing different r_{data} or equivalently z values) runs with both LTE and WLAN settings, for 10 different sample time periods taken randomly from the respective mobility dataset, yielding a total of 260 pairs of CL-FL experiments. All measurements of this section are taken at the end of each experiment and mean values out of the 10 samples are depicted together with the corresponding 95% confidence intervals.

Impact of r_{data} on the achievable testing accuracy

Both CL and FL successfully converge, achieving an average testing accuracy of 85% across both LTE and WLAN scenarios (refer to Fig. 2.4a). This result holds consistently for all values of $r_{data} \in [0.5, 15]$ (or equivalently $z \in [14, 426]$, where the x-axis is dual-valued, displaying dataset partitions z at the top and the parameter r_{data} at the bottom for clarity). The reported accuracy corresponds to the maximum value attained during the ML hyperparameter tuning phase (as illustrated in Fig. 2.3).

Our findings align with prior research, which highlights the importance of hyperparameter tuning and increasing the number of training epochs as effective strategies to enhance the convergence of FL's default algorithm, FedAvg [25]. This study further substantiates that properly configuring ML hyperparameters prior to training enables FL to achieve convergence across the entire range of $r_{data} \in [0.5, 15]$, ultimately attaining accuracy levels comparable to those of CL.

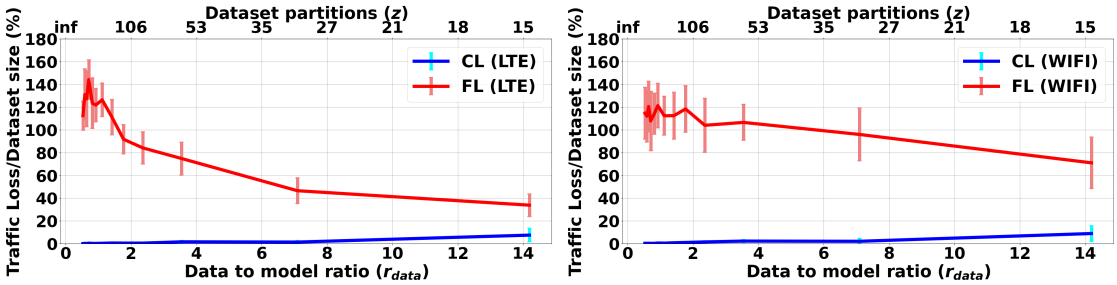


(a) Achieved testing accuracy (b) Normalized traffic overhead

Figure 2.4: Testing accuracy and normalized traffic overhead w.r.t. data to model ratio

Impact of r_{data} on the bandwidth consumption

The impact of r_{data} on the ML schemes' network resource utilization (reflected by the normalized traffic overhead) is depicted in Fig. 2.4b. We observe that for low ratios (r_{data}¡2), FL is bandwidth-demanding, consuming exponentially more data in comparison to CL. This stems from the fact that more clients participate in the training process, therefore more ML models are uploaded/downloaded to/from the server. In contrast, for large ratios (r_{data}¿3), where the total number of clients decreases, FL naturally becomes more bandwidth efficient. In fact FL demonstrates an exponential decrease of (exchanged) data compared to CL. This holds for both studied network settings, namely in LTE and WLAN. Note that this behavior is not related to the number of training rounds, since in our simulation training stops upon dataset depletion. Interestingly, a relative-equilibrium area exists when $2 < r_{data} < 3$, where CL and FL share similar network resource utilization profiles.



(a) LTE (b) WIFI

Figure 2.5: Normalized traffic loss w.r.t. data to model ratio

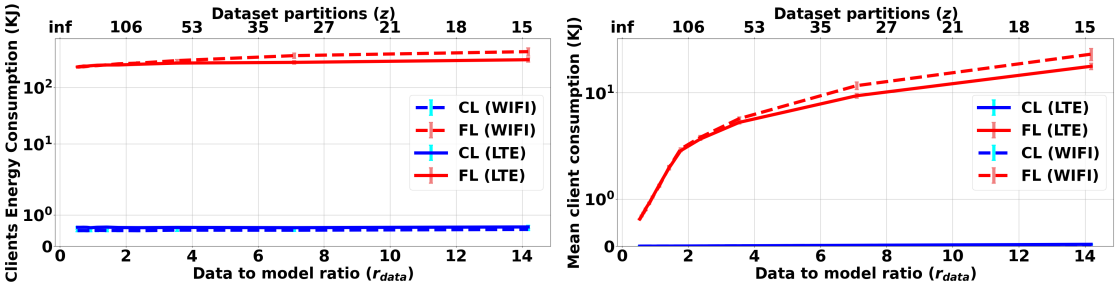
An increased r_{data} *i.e.*, a setup where few clients hold more data, would also benefit FL in terms of traffic loss, which basically represents unnecessary bandwidth usage from communication failures due to client mobility. Note that our setup does not focus on fairness or resilience aspects *e.g.*, replacing a failed node or investigation of asynchronous schemes [56]; it rather studies the effect of mobility on the default CL/FL setup, where failed nodes are excluded from the training/aggregation phases, respectively. As a result, performing FL with $r_{data} >> 3$ results in a traffic loss reduction up to 67% for LTE (see Fig. 2.5a) and up to 41% for WLAN (see Fig. 2.5b), as opposed to FL with $r_{data} < 2$. This reduction relates to the fact that in FL the client only exchanges ML models (of fixed size m) with the cloud server, regardless the actual amount of (raw) data he holds. Thus, a lower number of participating clients (for $r_{data} >> 3$) results in a lower number of exchanged ML models (*i.e.*, exchanged MBytes) and therefore decreased data loss. In FL (unlike CL) the traffic overhead and traffic loss accordingly is not governed by the size of each client dataset, but by the number of the total participants. Another point worth mentioning is that FL on a WLAN channel exhibits on average 40% more data loss compared to FL with LTE; at the same time, data loss in WLAN has larger confidence intervals compared to LTE. These observations reflect the average time a client is likely to remain (connected) in a service area; for WLAN that is by nature limited and erratic, as opposed to LTE, where users can potentially be always-on.

Contrary to the FL case, traffic loss is close to zero in CL, unless $r_{data} > 10$, where it reaches a maximum of 10%. In CL, no local processing (which is a time-consuming task in a resource-limited UE device) is required from the mobile clients; therefore delays may only occur due to transmission, which in turn minimizes the probability of a device

drop. On a more general remark for both ML schemes in LTE and WLAN, the traffic loss reaches a maximum of 16% of the total traffic overhead; this low percentage of losses is also reflected in the consistently almost-negligible confidence interval size of all graphs in Fig. 2.4b. Thus, the overall bandwidth expenditure (traffic overhead) appears (under the considered conditions) significantly more sensitive to the portion of data (r_{data}) each user holds, rather than his own mobility.

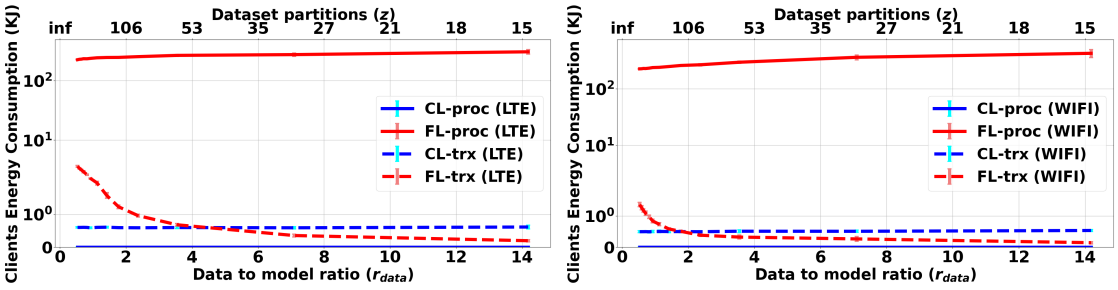
Impact of r_{data} on the energy consumption

Investigating the overall energy expenditure from the client's perspective, as analyzed in Sec. 2.3.5, our experimentation suggests that CL outperforms FL for all values of r_{data}. Specifically, if FL is employed the clients collectively consume 300 times more energy in the LTE case (that is 350 for WIFI), as opposed to CL (see Fig. 2.6a). The CL energy-efficiency (compared to FL) in the clients stems from the fact that it is the processing (*i.e.*, on-device ML training in FL) rather than data transmission which constitutes the prime factor for a UE's energy expenditure and thus its battery depletion. In fact, the total energy consumption in FL due to processing (proc) is 100 times higher (see Fig. 2.7a) compared to the total energy consumption due to transmission-reception (trx) in the LTE case (that is 280 times in WLAN - Fig. 2.7b). Since CL does not require local processing in the clients side, any minor gains in energy consumption due to transmission in FL (FL-trx) for larger r_{data} values (see Fig. 2.7) do not affect the overall ML schemes expenditure comparison. These minor gains stem from the fact that less ML models are exchanged in FL for higher r_{data} values.



(a) Total client energy consumption (b) Mean client energy consumption

Figure 2.6: (a) Total and (b) mean client energy consumption w.r.t. data to model ratio

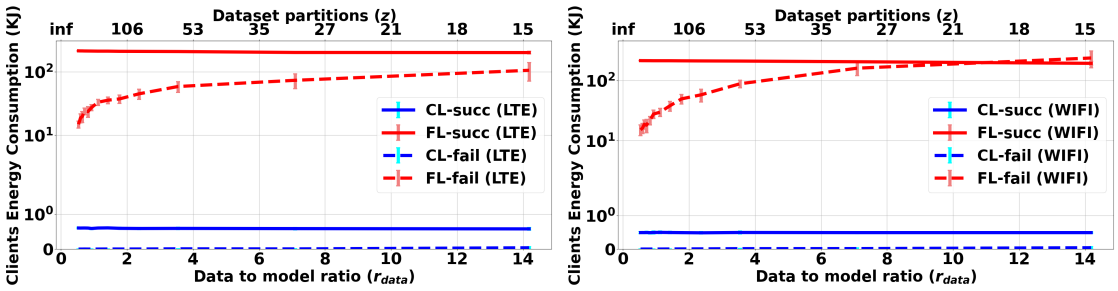


(a) LTE (b) WIFI

Figure 2.7: Client energy consumption w.r.t. data to model ratio, broken down to processing (proc) and transmission/reception (trx) for (a) LTE and (b) WIFI radio access technologies.

A secondary point to note is that the average per client energy expenditure *i.e.*, the total energy expenditure divided by the number of clients increases linearly as r_{data} increases (see Fig. 2.6b - note the logarithmic scale in y-axis). For our configuration, the mean client consumption varies in the interval $[0.64, 20.7]KJ$ for LTE and $[0.81, 28.8]KJ$ for WLAN. The fact that client consumption in WIFI is consistently increased compared to LTE stems from WLAN's lower data rates (see Sec. 2.3.1), combined with its higher energy transmission costs (see Sec. 2.3.5). Assuming a modern smartphone's typical battery [57] with 2400 mAh/3.8 V, resulting in 32.8 KJ battery capacity, we deduce that in the worst-case scenario of our configuration ($r_{data} \approx 14$), 63% of each UE's battery energy is depleted in the LTE case (that is 88% for WLAN).

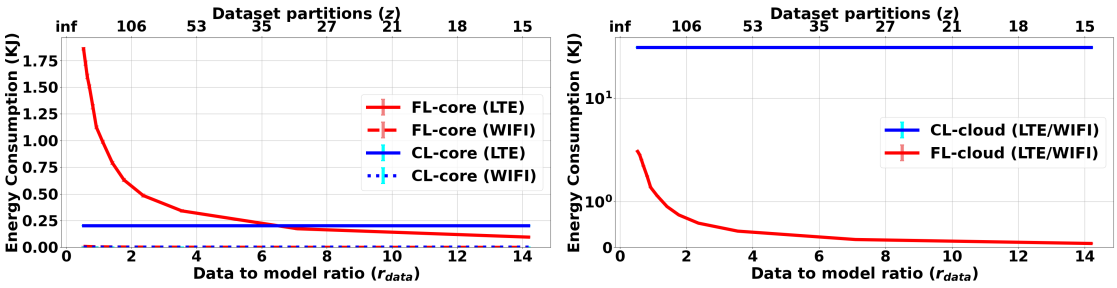
The results so far suggest that for $r_{data} > 4$, FL is on average 24% more energy efficient in an LTE setting, as opposed to WLAN (see Fig. 2.6). However, LTE's energy-efficiency does not appear in the CL case, although transmissions of large size datasets (instead of ML models) occur. As portrayed in Fig. 2.8, LTE's energy-efficiency is related to processing and specific to energy loss (see Sec. 2.4.1) *i.e.*, energy consumed for local processing, but followed by a communication failure. Since more communication failures occur in a WLAN setting compared to LTE, as discussed earlier in Sec. 2.4.3, the amount of Energy Loss (and thus total energy expenditure) is higher. In fact, energy loss is largely affected by r_{data}; larger r_{data} values represent larger per client dataset sizes implying higher energy consumption to perform training over these datasets. Thus, a potential communication failure of a client with a large r_{data} essentially means larger energy loss compared to a client with smaller r_{data}. In FL, energy loss amounts to less than 16% of the total expenditure for r_{data} <2, while reaches 30% for r_{data} >10 in the LTE case (see Fig. 2.8a). The respective WLAN values are 18% and 53% (see Fig. 2.8b).



(a) LTE (b) WIFI

Figure 2.8: Client energy consumption w.r.t. data to model ratio, broken down to successful (succ) and failed (fail) communication

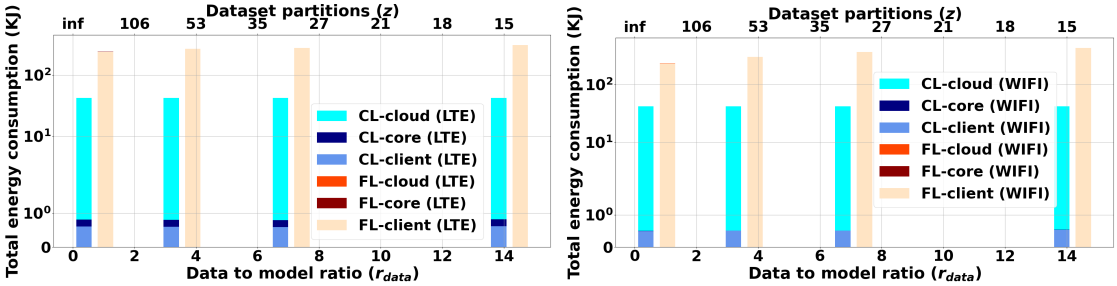
Unlike the client energy expenditure, both the core network's and the cloud's energy consumption exhibit an exponential reduction as r_{data} increases (see Fig. 2.9). For the core network, this reduction is only seen in the LTE case; the difference between the LTE-WLAN is mainly shaped by the energy expenditure in the network elements between the access and the metro & edge network, namely the BS and the AP respectively (see Sec. 2.3.1). Interestingly for the LTE case, a value exists for $r_{data} \approx 7$, where an equilibrium occurs between FL and CL (see Fig. 2.9a). In any case, core expenditure in LTE is at least two orders of magnitude greater compared to WLAN, regardless the ML scheme. As expected, the cloud's consumption is reduced, when the ML task is offloaded to the clients (see Fig. 2.9b); that reduction varies between 95% and 99%, as r_{data} increases from 0.5 up to 15 and relates to the total clients *i.e.*, the total (produced) ML models aggregated



(a) Core network (b) Cloud processing

Figure 2.9: Core and cloud energy consumption w.r.t. data to model ratio

in the cloud. The overall (system-wise) energy footprint for each ML scheme in the LTE and the WLAN case is depicted in Fig.2.10. Overall, CL demonstrates a higher energy efficiency compared to FL, both in LTE (see Fig.2.10a) and in WLAN (see Fig.2.10b). For $r_{data} \approx 1$, CL outperforms FL by 83% in the LTE case and 82% in WIFI. As $r_{data} \to 15$ these values are increased to 86% and 90% respectively. CL's energy efficiency is mainly dictated by the lack of processing (as opposed to FL) in clients' devices.



(a) LTE (b) WIFI

Figure 2.10: End-to-end energy consumption for all network stakeholders

Discussion on the effect of r_{data}

The overall performance of CL and FL as r_{data} increases is summarized in Table 2.3. A setting with reduced r_{data} values ($r_{data} \to 0$) *i.e.*, with many clients holding few data, greatly favors CL over FL. When r_{data} increases ($r_{data} \to \infty$), several benefits emerge for FL. FL becomes bandwidth-efficient, outperforming CL. As a result, clients (end-users) significantly reduce their data consumption, which is particularly advantageous in LTE environments where users often face constraints due to limited monthly data allowances. Additionally, the network infrastructure benefits from this reduction in data usage, as it alleviates the need for extensive bandwidth reservation and helps mitigate network congestion. An increased r_{data} also benefits both the network and the cloud infrastructure energy-wise. However, energy costs are distributed to the client devices, which can lead to client dropping, due to battery unavailability in a resource (battery)-constrained environment *e.g.*, in smartphones or IoT devices.

Client selection schemes offer a promising avenue for optimizing resource consumption across all network stakeholders, including clients, the network infrastructure, and the cloud infrastructure. These schemes can dynamically determine which clients participate in the

training process, factoring in criteria such as available bandwidth, energy constraints, and the quality of the local datasets. By carefully selecting participants, it is possible to balance resource usage efficiently, minimizing client energy expenditure and data transmission while also reducing strain on network resources such as bandwidth and server processing capacity.

For instance, prioritizing clients with high-quality datasets or stable connectivity could improve training efficiency and model accuracy while avoiding unnecessary overhead for clients with limited resources. Similarly, adaptive strategies that rotate client participation based on their resource availability and contribution potential could further enhance the overall system performance.

Exploring such client selection mechanisms could also pave the way for integrated optimization frameworks that address the trade-offs between energy consumption, latency, and model performance. These frameworks could even incorporate ML techniques to predict and adapt to dynamic network conditions, enabling a more intelligent and holistic allocation of resources. Investigating these aspects remains a rich area for future research, with the potential to significantly enhance the scalability and sustainability of distributed ML frameworks like FL.

Table 2.3: Effect of r_{data} on CL/FL performance

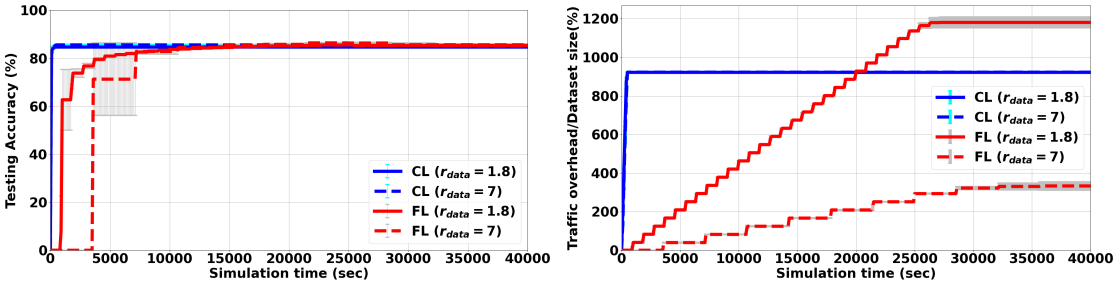
Metric w.r.t. r_{data}	CL	FL	Winner ($r_{data} \to 0$)	Winner ($r_{data} \to \infty$)
Achieved accuracy	Constant	Constant (via tuning)	None	None
Bandwidth consumption	Constant	Exponential decrease	CL	FL
Traffic loss	Constant	Linear decrease	CL	None
Total client energy consumption	Constant	Linear increase	CL	CL
Per client energy consumption	Constant	Linear increase	CL	CL
Energy loss	Constant	Linear increase	CL	CL
Core energy consumption	Constant	Exponential decrease	CL	FL
Cloud energy consumption	Constant	Exponential decrease	FL	FL

2.4.4 CL *vs.* FL: Convergence speed

In this section, we will analyse how ML schemes evolve across time. We make use of the samples (experiments) described in Sec. 2.4.3, zooming on the time dimension per experiment, instead of the final resulted values; for example Testing Accuracy refers to the achieved accuracy per time instance. Capitalizing on the results from the aforementioned paragraph, we limit our investigation to two values for r_{data}, namely 1.8 and 7, representing a setting with more clients which hold few data and a setting with few clients holding considerably more data, respectively.

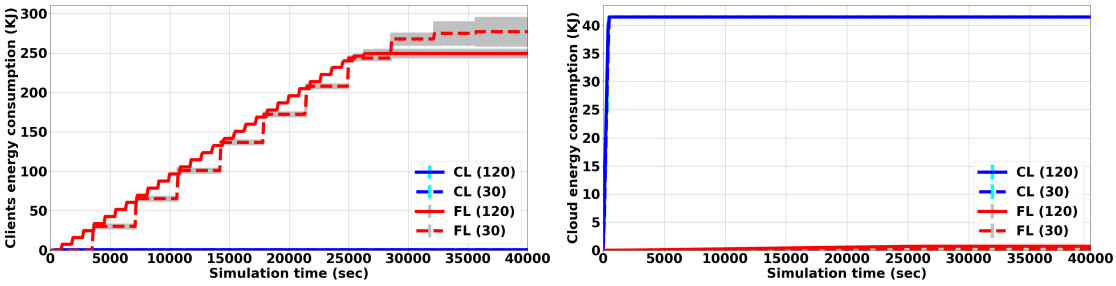
As depicted in Fig. 2.11a, CL reaches its maximum accuracy levels in the very first rounds (before 1K secs), utilizing the full dataset (see Fig. 2.11b). At that time it outperforms FL in terms of accuracy by an average of 22%. CL's faster convergence however, comes at a cost. In CL, data exchange is performed in a bursty manner, which results in a consumption of 100% of CL's required bandwidth before 1K secs. At the same time, the cloud's energy is consumed early compared to FL (42 KJ in less than 1000 secs), due to ML processing in the cloud server (see Fig. 2.12b). FL, on the other hand, proceeds gradually (linear evolution) towards its completion (at 13K secs benchmark for r_{data} =1.8 and 15K for r_{data} =7) and so does the network's data expenditure (see Fig. 2.11b) as well as the client energy consumption (see Fig. 2.12a). Since FL's accuracy reaches its peak before the end of the experiments (see Fig. 2.11a), any bandwidth or energy spent after the above-mentioned benchmarks has no obvious benefit. It is also observed that an increased r_{data} does not affect CL's behavior; in FL however, the ML task is performed

in a total of less client devices, therefore parallelism is decreased. As a result, the ML convergence, as well as the respective resource consumption footprint expands across time.



(a) Accuracy w.r.t. time (b) Bandwidth consumption w.r.t. time

Figure 2.11: Evolution of accuracy and traffic overhead across time



(a) Clients energy consumption w.r.t. time (b) Cloud energy consumption w.r.t. time

Figure 2.12: Evolution of energy consumption in the (a) clients- and (b) cloud-side across time

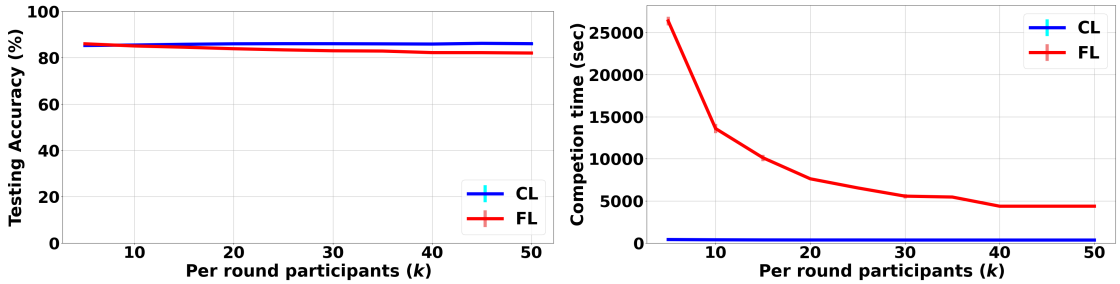
Overall, unless bandwidth constraints are a critical factor within the system, CL emerges as the preferred option for accelerating the training process. This makes CL particularly well-suited for ML applications where data evolves rapidly over time, such as time-series forecasting. By leveraging CL's faster completion times, it is possible to continually produce up-to-date global ML models, ensuring relevance and accuracy in dynamic scenarios. Additionally, the faster execution of CL reduces the duration for which network resources are occupied, freeing capacity for other applications and mitigating the risk of network bottlenecks.

Conversely, FL offers distinct advantages when a gradual, balanced approach to achieving maximum ML accuracy is prioritized, especially in scenarios where conserving network resources is more critical than minimizing time, such as in data analytics applications. FL's distributed nature inherently economizes bandwidth by avoiding the transmission of raw data, which can be advantageous in resource-constrained environments. Furthermore, FL's efficiency could be enhanced through the implementation of a mechanism that continuously monitors accuracy improvements during training. Such a mechanism would identify when convergence is achieved, allowing the training process to halt, thereby avoiding unnecessary resource consumption.

To maximize the strengths of both approaches, a dynamic and adaptable system akin to Hybrid Learning [18] could be employed. This hybrid framework would intelligently switch between CL and FL based on the specific requirements of the ML application and the current availability of system resources. Such a solution could optimize both

performance and resource utilization, ensuring that the training process aligns seamlessly with the constraints and objectives of the operational environment.

2.4.5 CL *vs.* FL: Varying the number of participating clients



(a) Resulted accuracy w.r.t. per round participants

(b) Task completion time w.r.t. per round participants

Figure 2.13: Impact of increasing the number of participating clients per round

We now fix the number of clients/partitions to $z = 100$, or $r_{data} \approx 2$, an area where CL and FL demonstrate a similar bandwidth consumption profile; moreover, FL exhibits certain gains in terms of the client energy consumption (see Sec. 2.4.3). Our goal is to investigate how each ML scheme's accuracy is affected, when the number of participating clients per round k varies. Essentially, a small value of k represents a setting where few clients are selected in each round *e.g.*, due to low availability or client scarcity, therefore more communication rounds are required to complete the ML task. Vice versa, an increased k suggests a client-rich setting, where more clients participate in each round. We vary k from 5 clients per round (which was our initial setting) up to 50, with a step of 5. Each experiment is repeated for 10 different time periods/samples, taken from the LTE mobility dataset, resulting in a total of 100 pairs of CL-FL experiments.

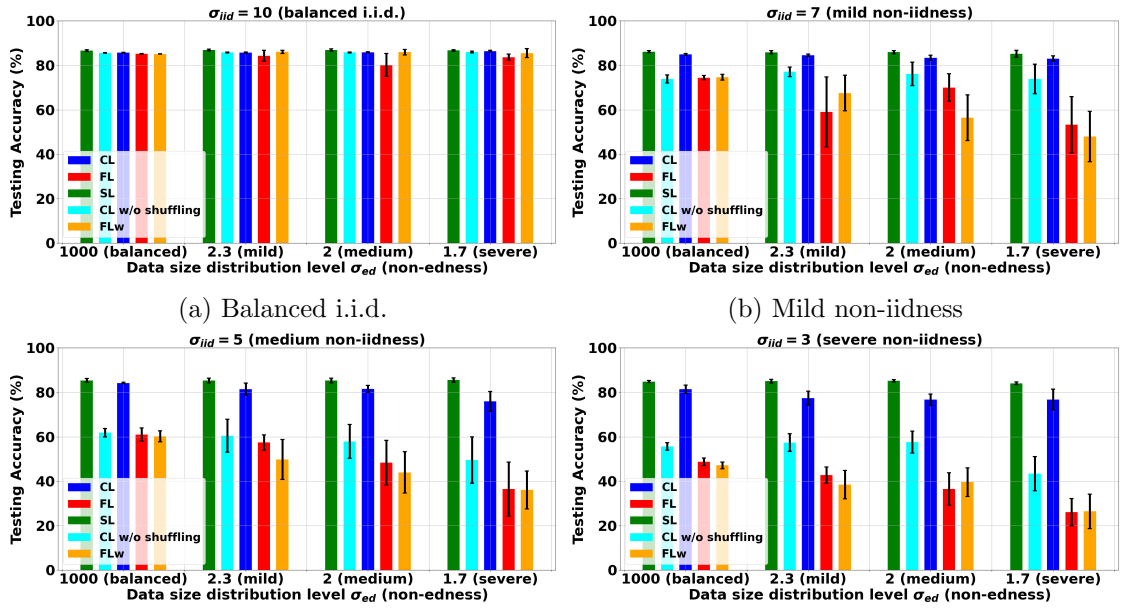
The results suggest that scaling the participants has practically no effect on CL's accuracy (see Fig. 2.13a) nor on its completion time (see Fig. 2.13b). FL's accuracy on the other hand is reduced up to 4%, when k increases. This result is in line with [20], [58], where it is shown that increasing parallelism (*i.e.*, allowing for more per round participants and decreasing the total rounds) degrades the overall accuracy, due to the smaller number of aggregation (FedAvg) steps. In fact, it is suggested that an optimal k value exists in FL's client selection process, depending on the total number of online clients, the selected ML hyperparameters and the nature of the ML task. For image classification tasks like SVHN with a batch size of 128, the ratio of k/z needs to be close to 0.1.

Interestingly, FL enjoys a nearly-exponential reduction in completion time, as k (linearly) increases *i.e.*, an increase of k from 5 to 50 reduces the completion time by 83% (see Fig. 2.13b). The relatively small number of extra participating clients *i.e.*, no more than 50, should be easy to realise in a real-world system of hundreds of users *e.g.*, mobile devices in an urban cellular network. If $k \to 50$, the time gap between the completion times of CL and FL is significantly reduced, although FL experiences some limitations in terms of accuracy. This trade-off is absent in CL, where scaling the number of clients does not have a notable effect on the performance or the training process. An important observation relates to the impact of scaling on network resource consumption. As detailed in Sec. 2.4.3, the relevant resource consumption metrics increase in a consistent, proportional manner without any anomalies. Therefore, we have chosen to omit the corresponding plots, as

they do not provide additional insight.

2.4.6 CL *vs.* FL: Effect of data heterogeneity

Thus far the SVHN dataset was assumed to be evenly distributed (in size) across the clients (e.d.) and that each client's subdataset is representative compared to SVHN (i.i.d.). We now investigate the ML schemes' response (in terms of achieved accuracy), when the above-mentioned assumptions are relaxed and furthermore shed some light on the interplay between *non-iidness* and *non-edness* (see Sec. 2.3.3). For the non-e.d. setting, we assume four cases for the Zipf parameter $\sigma_{ed} \in \{1.7, 2, 2.3, 1000\}$, representing various degrees of data distribution skewness; as $\sigma_{ed} \to 1$, the skewness degree becomes severe. Similarly for the non-i.i.d. setting, we select four cases for the i.i.d. parameter $\sigma_{iid} \in \{3, 5, 7, 10\}$, which represent the total number of classes contained in a client's subdataset. SVHN contains samples from all 10 classes, thus a $\sigma_{iid} = 10$ is essentially an i.i.d. setting, while smaller σ_{iid} values mark increasing non-i.i.d. conditions. We also assume a total number of clients $z = 100$ and fix the per round participants $k = 10$, as suggested by the results of Sec. 2.4.5. For comparison purposes, we include two extra centralized ML schemes besides CL *i.e.*, Standard Machine Learning (SL) and CL without (w/o) shuffling and one additional federated *i.e.*, Federated Learning with weighted averaging (FLw). For SL, the server waits for the majority (at least 80%) of SVHN data to arrive and then performs training. CL without shuffling is identical to CL, however no data shuffling occurs prior to training. FLw uses a weighted averaging scheme during aggregation, as opposed to FL's uniform aggregation (see Sec. 2.3.6). Each experiment is repeated 10 times, using LTE traces, resulting in a total of 160 sextuples of {SL, CL w/o shuffling, CL, FL, FLw} experiments.



(a) Balanced i.i.d.

(b) Mild non-iidness

(c) Medium non-iidness

(d) Severe non-iidness

Figure 2.14: Effect of client data size (e.d.) and content (i.i.d.) variations

From all centralized ML schemes (SL, CL and CL w/o shuffling), SL exhibits the highest performance, reaching an accuracy of 85% (see Fig. 2.14). SL's performance is not affected by the various degrees of non-edness or non-iidness, essentially reflecting the advantages of centralized ML under heterogeneous environments [25]. In SL, the cloud

server waits for almost all subdatasets to arrive, before merging them into a super-dataset and initiating the ML training. Therefore, any stochasticity introduced in the (distributed) client subdatasets, either in size or content, is eliminated when the merging occurs. As discussed in Sec. 2.3, SL is impractical in a real system, so it is merely used for comparison purposes.

CL w/o shuffling can be regarded as a dynamic alternative to SL, since training is performed in each communication round, using the collected round's client data. Under an i.i.d. setting (see Fig. 2.14a), CL w/o shuffling achieves similar performance to SL, which is not affected by the data size distribution (non-edness). When non-iidness appears and as it increases (see Fig. 2.14b-Fig. 2.14d), CL w/o shuffling exhibits lower accuracy levels (down to 56%). The introduction of non-edness further diminishes its performance, which drops down to 44% for severe non-i.i.d. and non-e.d. conditions (see Fig. 2.14d). This degradation is related to the distributed manner of client data and the fact that ML training is performed in each round. Under high data heterogeneity, consecutive series of non-representative data samples can be uploaded in the cloud server *e.g.*, several batches containing only one SVHN class out of ten. Such bias is likely to have a detrimental effect on the ML training task. Moreover, given that each round's ML model is used for the next round training, any abnormalities will be cascaded, resulting in less accurate ML models. To mitigate those negative effects, we apply and experimentally evaluate a popular technique prior to ML training *i.e.*, shuffling [53]. CL with shuffling is referred to as CL, for simplicity. As portrayed in Fig. 2.14, CL achieves the same or higher accuracy levels compared to CL w/o shuffling. In fact, SL only outclasses CL by a maximum of 2% for mild (see Fig. 2.14a-2.14b) and 7% for major (see Fig. 2.14c-2.14d) data heterogeneity respectively.

FL on the other hand, being a distributed learning scheme is affected both from non-iidness and non-edness; even the introduction of small levels of non-iidness can lead to accuracy drops down to 53% (see Fig. 2.14b). When in fact non-iidness is combined with non-edness, the training task cannot converge (see Fig. 2.14d), therefore accuracy drops below 40%. Such performance is related to FL's training algorithm (uniform FedAvg). As such, a ML model trained with high diversity data will be equally weighted with another model (from another client) trained with low diversity (biased) data. FLw (FedAvg with weighted averaging) introduces a simple bias mitigation technique by rewarding ML models trained with larger datasets with larger weights. When non-iidness is low, this technique enables FLw to achieve an up to 11% better accuracy compared to FL for various levels of non-edness (see Fig. 2.14a-2.14b). However, under the presence of higher levels of non-iidness (see Fig. 2.14c-2.14d) FL achieves similar or (up to 8%) better accuracy compared to FLw. This is because weighted averaging (FLw) rewards ML models according to their training data quantity (larger datasets have larger weights) and not quality *i.e.*, data diversity, entropy, number of classes in the dataset As such, not only FLw is prone to non-iidness (similarly to FL), but can potentially demonstrate erratic behavior [59]. Overall, it becomes evident that applying FL over highly heterogeneous data environments requires advanced statistical methods that deserve a dedicated exploration.

2.5 Conclusion

This chapter has presented a comprehensive examination of the impact that employing ML pipelines in networked environments has on the underlying network resources. Our investigation was grounded in an end-to-end, systematic analysis enabled by a novel, measurement-based, pragmatic model. This model accounts for the network resource

consumption associated with CL and FL when applied to image classification tasks. To ensure real-world relevance, the model integrates user mobility patterns derived from actual traces to capture environmental dynamics. Furthermore, it incorporates multiple communication channels (LTE and WLAN) to facilitate comprehensive experimentation. Our findings, derived from simulations conducted with the SVHN dataset using openly shared simulation code, are summarized below.

Firstly, we highlighted the necessity of hyperparameter tuning before initiating ML training, particularly emphasizing the importance of configuring the number of local training epochs. This is crucial to ensuring FL convergence, especially when clients possess local datasets smaller than twice the size of the ML model. By increasing the number of local FL epochs, we demonstrated that FL could achieve accuracy levels comparable to CL, albeit with a proportional increase in client energy consumption. When clients hold larger datasets relative to the ML model size, FL offers several advantages over CL. These include: (1) an exponential reduction in bandwidth usage, accompanied by decreased energy consumption in the cloud and core network (notably over LTE); and (2) a reduction in data loss due to mobility by at least 40%.

In terms of energy consumption, we observed that ML processing on FL clients significantly outweighs the energy demands of data transmission, regardless of the communication channel (LTE or WLAN). For clients with local datasets exceeding 10 times the size of the ML model, battery depletion becomes a significant concern, with at least 63% of an average user equipment's battery consumed. This high energy demand may deter client participation in FL, particularly in scenarios aimed at alleviating network congestion.

When comparing convergence rates, CL proved to be 13 times faster than FL, making it an appealing option for minimizing the duration of ML tasks assigned to system devices. However, this accelerated convergence comes at a steep cost, including spikes in both energy consumption (within the cloud) and bandwidth usage. FL, in contrast, offers a more gradual convergence process that is considerably more bandwidth-efficient while still achieving similar accuracy. Consistent with prior studies, we confirmed that increasing the number of participating clients per FL training round accelerates convergence at the expense of a marginal reduction in final accuracy.

Finally, we examined the effects of data asymmetry on the accuracy of ML models. In FL, such asymmetry—whether due to variations in client dataset sizes or content—significantly degrades accuracy. Conversely, in CL, this issue can be mitigated through pre-training data shuffling techniques, which standardize data prior to centralized ML training.

This research opens the door to several promising avenues for future work, many of which can be explored using our publicly available simulation software. For instance, the relationship between the training data-to-model ratio and ML performance warrants further investigation, particularly with more complex and larger neural network architectures. Other factors such as data acquisition rates and client storage limitations could also enhance the realism of future studies. Additionally, the exploration of algorithms designed to mitigate bias in non-iid FL datasets remains a critical area for improvement. Lastly, scrutinizing dynamic ML paradigms, such as Hybrid Learning, could yield valuable insights into the parameters influencing the selection of ML schemes in resource-constrained network environments. Through this study, we have not only characterized the trade-offs between CL and FL but have also laid the groundwork for advancing ML deployment strategies in dynamic, resource-limited settings.

3

Federated Learning in the course of time

3.1 Introduction

In the previous chapter, we conducted a comprehensive end-to-end system comparison between the traditional CL approach and the emerging paradigm of DML, focusing specifically on FL as a typical example of DML. In FL, the model training task is collaboratively performed by the clients while preserving data privacy [60]. Our exploration compared the performance of CL and FL in terms of accuracy, while also accounting for the resource consumption of the involved stakeholders in the underlying network, such as client devices, the network infrastructure, and cloud/edge servers.

ML tasks that take place within dynamic network environments often involve mobile and distributed client devices, such as smartphones, OBUs, and wearables. These devices are typically subject to mobility and can be considered part of applications such as Cooperative, Connected, and Automated Mobility (CCAM) services [61], mobile network analytics [62], and e-Health applications [63]. In these settings, data is often collected progressively over time—spanning days, weeks, or even months—contrary to traditional ML paradigms that assume the availability of a complete dataset prior to training. This progressive data collection has led to the development of Lifelong Learning (LL), which enables ML models to be trained periodically and used for inference in the course of time [64], [65].

A critical challenge that arises in such environments is concept drift, which refers to changes in the statistical properties of the data over time due to factors such as seasonality, evolving user behavior, and trends [66]. If left unaddressed, concept drift can lead to model drift, resulting in a degradation of the model's accuracy. In traditional CL, where data is centrally collected, concept drift is typically managed through statistical techniques applied directly to the raw data [67]. However, in distributed environments such as FL, significant limitations hinder the application of centralized solutions: 1) the FL server does not have access to raw client data and thus cannot directly monitor changes in data distributions, and 2) client devices are often resource-constrained (*e.g.,* limited processing power, storage, and battery life) and can be unreliable (*e.g.,* due to connection failures or dropouts), making them ill-suited to handle drift detection and mitigation compared to a stable, always-on server. These challenges give rise to two fundamental research questions: 1) *How can concept drift be accurately detected in a Federated Learning environment, given the aforementioned restrictions?* and 2) *How can the resource consumption cost associated*

with drift mitigation (e.g., re-training) be minimized, particularly in resource-constrained client and network settings?

To the best of our knowledge, there have been only a few studies addressing drift management in the context of Federated Learning [68], [69]. However, existing solutions often either *violate* the data privacy principles of FL or propose *continuous* training approaches that lead to excessive energy and bandwidth consumption—an issue we explore in this chapter. To address this research gap, we propose a novel, resource-efficient drift management solution for Federated Learning, called the Drift-aware Resource-efficient Federated Learning (DareFL) algorithm. This algorithm is specifically tailored to mobile and vehicular networks, which are often resource-constrained. DareFL integrates existing centralized statistical drift detection techniques [66] while maintaining FL's strict data privacy requirements. Upon detecting drift, the system mitigates the impact by re-training the affected ML model. The periods for re-training are carefully controlled, ensuring that the model remains accurate while minimizing resource consumption for the underlying network.

In contrast to most existing drift management solutions, which are typically evaluated on limited datasets for niche applications (*e.g.,* digit recognition [65]), we place a strong emphasis on realistic evaluation environments. Specifically, we focus on the predictive Quality of Service (pQoS) use-case [70], which is an automotive ML task operating in distributed, mobile environments prone to concept drift [71]. The pQoS use-case involves predicting changes in network QoS in advance, enabling timely actions to be taken by applications running on client devices or application servers. For example, the system could trigger an automated handover of control from an autonomous vehicle to the driver to avoid critical situations.

To overcome the challenge of limited public pQoS data, we generate synthetic datasets through a combination of network and traffic co-simulation, utilizing real-world maps. These datasets capture two distinct drift scenarios: one related to the dynamics of wireless communication and the other to variations in user behavior. The generated pQoS datasets are used in a distributed ML simulator that emulates the training process, while also capturing key aspects of energy and bandwidth consumption based on real-world measurements and commercial product benchmarking.

In our experiments, we evaluate the performance of the proposed DareFL algorithm in the presence of concept drift, comparing it against three baseline approaches: a) Vanilla Federated Learning [72], b) a continuous training scheme, and c) a representative state-of-the-art (SotA) drift-mitigation solution [69]. The remainder of the chapter is organized as follows: Related work is discussed in Sec. 3.2. The system architecture is presented in Sec. 3.3, followed by the description of the simulation framework in Sec. 3.4. In Sec. 3.5, we present a simulation-based performance evaluation of our proposed solution. Finally, a summary is provided in Sec. 3.6, along with a discussion on potential directions for future research.

3.2 Related work

The problem of drift management—comprising both *detection* and subsequent *mitigation*—has been widely explored in the context of CL, where datasets, training nodes, and resulting ML models are typically co-located in a central location. Drift detection within CL environments is often tackled using two primary categories of detectors: *data distribution-based* and *performance-based* methods [66]. The former monitors the distribution of training data, comparing it against historical data to identify potential drifts.

These methods rely on statistical tests, such as Kullback-Leibler divergence [67], to quantify the changes in data distributions over time. By identifying discrepancies between the current training data and previous data distributions, these methods can flag concept drift.

On the other hand, *performance-based* drift detectors focus on tracking shifts in the model's inference performance, particularly by monitoring the test error associated with predictions. These approaches are grounded in the Probability Approximately Correct (PAC) concept [66], which postulates that any significant degradation in a model's performance signals that the relationship between the input data and the target output has changed. This change implies that a concept drift has occurred, requiring immediate intervention. Upon detection of concept drift, mitigation strategies are employed, which typically involve re-training the models using updated data, combining both old and new models, or fine-tuning the existing models based on newly acquired information [67].

In the context of distributed settings, such as FL, drift management is a more complex challenge due to the decentralized nature of the training process. Three main strategies have been explored to address concept drift in FL: (a) Personalized learning, (b) Asynchronous FL schemes, and (c) Continuous Federated Learning techniques.

Personalized learning

It aims to tailor a global model to individual clients, adapting it to their specific needs or data distributions. This method enables clients to re-train a generic global model to produce customized, client-specific models. For instance, Jothimurugesan et al. [73] proposed a mechanism where clients detect drifts locally by monitoring their own data streams. Clients experiencing concept drift are then grouped by the server, and each group is assigned a distinct FL model that is fine-tuned to their data. Other approaches propose maintaining both a global and a local client model, allowing clients to select the model that best fits their data [74].

In some cases, clients may even rely on meta-data collected from other clients to inform their model choices [75]. Additionally, a more generalized approach is to enable clients to retrieve multiple personalized models from the central server using a publish-subscribe mechanism, where the clients perform model aggregation locally [76]. However, while personalized learning offers a flexible solution, it significantly increases system complexity, particularly in large-scale FL environments with thousands of clients. Managing multiple models for each client can strain the available computational and storage resources, presenting a challenge in terms of scalability.

Asynchronous Federated Learning

This method takes a different approach by allowing clients to independently train and upload their models when necessary, particularly when a drift is detected. Drift detection in Asynchronous FL typically occurs at the client-side, where local data is compared to historical data [60], or by assessing the changes in the global model's performance [77], [78]. Once drift is detected, mitigation strategies are applied at the client level. These strategies may involve re-training the local model [60], employing ensemble methods [77], or adapting the local cost function [78]. While asynchronous techniques offer flexibility and efficiency in addressing concept drift, they introduce additional complexity in terms of computational resources, storage, and synchronization overheads. These challenges are especially pronounced in resource-constrained environments, where clients may lack the capacity to handle the increased demands of frequent model updates and drift detection.

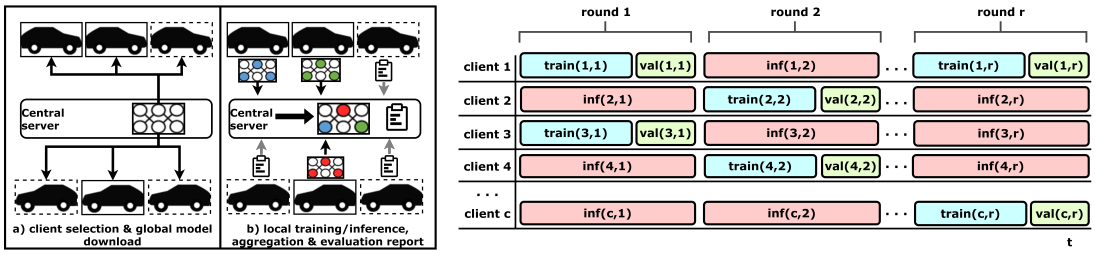


Figure 3.1: Vanilla FL framework

Continuous Federated Learning

An alternative solution to managing drift in FL is Continuous Federated Learning (ConFL), which involves repeatedly re-training the ML model to adapt to emerging drifts over time. In this approach, clients continuously monitor their data streams for signs of drift, and when drift is detected, they trigger re-training or adjustment of the global model. Manias et al. [68] explored a technique where clients detect drifts using Euclidean distance metrics, and the server isolates those clients experiencing drift from the training process. However, this method falls short in the case of a global drift that affects all clients, as it relies on local detection and does not account for shared, global drifts across the network.

A more generalized approach involves using convex optimization techniques to detect drift server-side, as shown in [79], where the server adapts the number of training epochs in response to drift. However, this approach has been primarily demonstrated in near-stationary environments, where the data distribution does not fluctuate too rapidly. Similarly, AdaptFL [69] detects drift by comparing client model parameters received at the server-side, using a moving average approach. Once drift is detected, the server adjusts the learning rate for the next training round and communicates this adjustment to the clients. The learning rate is increased in the presence of drift and gradually decreased otherwise. While both ConFL and AdaptFL have demonstrated success in mitigating concept drift, they rely on constant re-training, which, in the absence of drift, consumes excessive network resources without yielding significant gains in model accuracy.

Overall, drift detection and mitigation in distributed environments, especially in FL, pose significant challenges due to the decentralized nature of training and the diverse data sources involved. While various methods have been proposed to address these issues, each approach comes with its own set of trade-offs, balancing accuracy, resource consumption, and system complexity. As the field of federated learning continues to evolve, further exploration is needed to develop more efficient, scalable, and resource-conscious methods for handling concept drift in large-scale, dynamic environments. The ongoing research in this area has the potential to significantly enhance the robustness and effectiveness of federated learning systems, particularly in real-world applications where data distributions are continuously changing.

3.3 System architecture

3.3.1 Vanilla Federated Learning

Firstly, we introduce the foundational concepts that underpin Vanilla FL in distributed environments, over the course of time. We consider a typical scenario involving a centralized server, such as one hosted in a cloud infrastructure, and a set of mobile clients,

for instance, vehicles. These clients are equipped with the following capabilities: a) raw data acquisition, typically through sensors; b) processing capacity to execute (ML) training, facilitated by local processors; and c) connectivity to the central server, typically via cellular or other wireless networks.

Model preparation (server-side)

The process begins with the initialization of a custom ML model at the server, which, at the start, is untrained (depicted as a blank model on the leftmost part of Figure 3.1). To ensure optimal performance during the subsequent training phase, a "warm-up" period is introduced. This phase consists of a series of test runs on the server, using a limited dataset to calibrate the model. The warm-up period involves two critical stages: model design and hyperparameter tuning. During the design phase, the structural components of the model are defined, including the layers, loss function, activation function, and learning rate scheduler [80]. Subsequently, hyperparameters such as batch size, learning rate, and number of training epochs are optimized using techniques like grid search over predefined ranges [81].

Data pre-processing (client-side)

The raw data generated by each client undergoes a series of pre-processing steps before being used for training. These steps may include data cleaning, augmentation, normalization, and feature extraction [80]. After pre-processing, the data is partitioned into two subsets: a training dataset and a validation dataset. The training dataset is used to update the ML model, while the validation dataset is employed to monitor the training process, ensuring that overfitting is prevented through techniques like early stopping [80]. Additionally, both datasets undergo normalization to ensure that the model handles the data efficiently and effectively [81]. Common normalization methods, such as Min-Max scaling or Max-Abs scaling, are applied [80]. Once the model has been trained on the client's data, it is then used for inference, evaluating the model's performance on unseen data.

Federated training and testing in the course of time

Training in the Vanilla FL framework occurs in successive cycles, or training rounds R, each involving a set of clients. For each round $r \in [1, R]$, the central server randomly selects a subset of K clients to act as trainers and M clients to act as testers, where $K + M \leq C$ represents the total number of clients C in the system [82]. The global ML model is distributed to the selected trainers and testers, and each trainer uses its local data to train the model, resulting in the generation of a local model. These models are depicted in various colors (green, blue, red) for different clients in Figure 3.1. In parallel, each tester uses its data to infer the performance of the global model on the current round's dataset.

Upon completion of the training phase, the server collects the local models and aggregates them into a new, updated global model using the Federated Averaging (FedAvg) algorithm [69]. The server also collects the inference results from the testers, which are used to generate a performance evaluation report for the global model. This evaluation does not involve any disclosure of client data or statistics, ensuring the privacy of client information, which is a key feature of FL.

At the conclusion of round r, the updated global model is re-distributed to a new set of K trainers and M testers for the next round $r + 1$, and this process continues until

the termination criteria are satisfied. Termination may occur upon reaching a predefined maximum number of rounds, achieving a global model accuracy above a certain threshold, or other criteria [72]. Training is organized into rounds of fixed duration q. While asynchronous approaches to training could be considered [78], they are susceptible to challenges such as communication bottlenecks and client-induced bias. Clients continuously collect data at a frequency f, but only the data gathered within each round's duration q is used for training. Consequently, each selected trainer $c \in [1, K]$ divides its collected data into a training dataset, denoted as *train*(c, r), and a validation dataset, denoted as *val*(c, r). Likewise, each tester $c \in [1, M]$ generates an inference dataset, denoted as *inf*(c, r), as depicted in light blue, light green, and light red in Figure 3.1.

3.3.2 Concept drift detection and mitigation

Concept drift in ML refers to the phenomenon whereby the statistical properties of data change over time in unforeseen ways. Assuming a distribution P_t between the independent variables X and the dependent variable y, across time t, drift occurs if $\exists t : P_t(X, y) \neq P_{t+1}(X, y)$. This definition assumes a single data distribution, although each FL client can experience different drifts.

Our study operates under the assumption of a global drift, which is induced by a widespread change in the environment. In this scenario, all clients are impacted by the drift, although the severity or intensity of the drift may vary across different clients [60]. VanillaFL, as described in Sec. 3.3.1, is inherently ill-equipped to handle such drift effects. In this setting, training halts once the system reaches convergence, which poses a significant challenge. Any subsequent changes in the client datasets—such as shifts in data distribution or emerging patterns—can lead to model degradation, as the model fails to adapt to these changes after training is completed [66].

To address this limitation, state-of-the-art variations of Vanilla FL, such as Continuous FL and Personalized Learning, have been proposed. While these adaptations attempt to mitigate the effects of drift, they come at a cost. Specifically, they tend to lead to increased resource consumption or add significant complexity to the system (as detailed in Sec. 3.2). ConFL requires continuous re-training of the model, which demands substantial computational and communication resources, particularly in large-scale environments. On the other hand, Personalized Learning, while capable of tailoring models to individual clients, increases the overall system's complexity by necessitating the maintenance of multiple models per client. These trade-offs highlight the challenges of efficiently managing concept drift in federated learning systems, particularly when balancing accuracy, resource consumption, and system scalability.

In contrast to the aforementioned methods, our approach centers on the precise detection of two key events: 1) the completion of the training process, which signifies convergence, and 2) a shift in the underlying data distribution, known as concept drift. By detecting these events, we can effectively halt or (re)initiate the training process in a timely manner, ensuring the model adapts appropriately to changes. The challenge in FL arises from the fact that clients are generally unable to share their individual datasets, which limits the ability to directly monitor data changes or model performance. However, clients can provide performance indicators derived from their local ML models, such as inference results, which can be used as proxies for assessing the model's effectiveness.

The core concept of our approach lies in the use of performance-based detection mechanisms. These detectors are based on the assumption that a degradation in the ML model's performance is indicative of issues such as overfitting or the onset of concept drift. The

PAC model, referenced in Sec. 3.2, forms the theoretical foundation for this detection. In the case of overfitting, the model has already converged, but continued training leads to diminishing returns, with the model fitting noise or irrelevant features. In contrast, concept drift reflects changes in the data distribution, meaning that the model's previously learned relationships no longer hold true, leading to a drop in accuracy. By monitoring performance locally at each client, we can detect these phenomena and make informed decisions about when to stop or restart the training process, ensuring that the model remains effective and responsive to data changes.

Depending on the specific ML task, a variety of absolute performance metrics can be employed to assess the accuracy of predictions relative to the ground truth. These metrics may include Accuracy (%) for classification tasks, Root Mean Square Error (RMSE) for regression, the Silhouette Coefficient for clustering, among others [80]. While these absolute metrics are useful for evaluating model performance in stable environments, they can lead to misleading conclusions, particularly in highly dynamic settings. In such environments, the performance of an ML model can degrade for a variety of reasons, including local client biases, the presence of noise, or changes in the data distribution, all of which can negatively impact the validity of the performance assessment [83].

To mitigate the limitations of using absolute metrics in these contexts, we propose a novel approach that incorporates a custom performance indicator. This indicator compares the ML model's current performance against a baseline set by a naive algorithm, which is determined locally by the clients. Naive algorithms, such as random selection for classification tasks, rolling means for regression, or other simple heuristics, are commonly used as benchmarks in ML research. These algorithms provide a lower bound for model performance, indicating the minimum level of accuracy or performance that the model should exceed. By comparing the current model performance to this baseline, we can more accurately assess whether the model's predictions are still meaningful and reliable, even in the presence of changes in the data distribution, noise, or other disruptive factors. This approach allows for a more robust detection of concept drift and performance degradation, offering a practical solution for maintaining the effectiveness of ML models in dynamic environments [81].

Our approach offers several notable advantages. First, it incurs negligible computational cost, as the naive algorithm only requires minimal computations, especially when compared to the intensive processing involved in running machine learning models. This makes the approach highly efficient, particularly in resource-constrained federated learning environments where clients may face limitations in computational power. The simplicity of the naive algorithm means it does not place a significant burden on client devices, making it an ideal solution for frequent, low-cost performance monitoring.

Second, our approach provides a more reliable estimation of the machine learning model's accuracy. Unlike absolute performance metrics, which can be influenced by various external factors such as local data biases or noise, our method uses a relative comparison. By measuring the ML model's performance against a naive baseline, we obtain a clearer picture of the model's effectiveness. This approach is more robust in highly dynamic environments, as it helps mitigate the inaccuracies that may arise when absolute metrics alone are used. The relative comparison enables a better understanding of whether the model is still performing optimally or if its accuracy has declined due to issues like concept drift or overfitting.

In practice, for a given set of inference samples, the client conducts two types of inference: one using the trained ML model and another using the naive algorithm. The accuracy for each inference is then calculated using an absolute metric, such as accuracy for

classification tasks or RMSE for regression. Finally, the two accuracy values are compared with one another yielding a single indicator value, denoted as *kpi*. *kpi* expresses the ML model's performance enhancement over the naive algorithm. If ACC_{ml} and ACC_{naive} mark the absolute metrics of the ML's models and the naive algorithm's performance respectively, our indicator *kpi* of a client c in a round r becomes:

$$kpi^{c,r} = 100 \cdot (ACC^{c,r}_{naive} - ACC^{c,r}_{ml})/ACC^{c,r}_{naive} \tag{3.1}$$

3.3.3 The Drift-Resilient Resource-Aware (DareFL) algorithm

Algorithm 1 *DD*(**Input** : {*kpi*})

```
define DDM list: {ddm}
for each element e ∈ {kpi} do
    if e >= β1 then
        append 0 to {ddm}
    else
        append 1 to {ddm}
    end if
    return DDM({ddm}, β2, β3)
```

Algorithm 2 *CD*(**Input** : {*kpi*})

```
define ckpi list: {ckpi}
ckpi=mean({kpi})
append ckpi to {ckpi}
if {ckpi}
    not increase(β4) then
    return boolean=True
else
    return boolean=False
```

We propose a novel algorithm, Drift-aware Resource-efficient FL (DareFL), which is designed to address two key objectives. First, it ensures timely halting of the training process once model convergence is achieved, thus preventing unnecessary resource consumption. This early termination not only saves computational resources but also optimizes energy consumption and reduces network communication costs. Second, DareFL incorporates an effective drift detection mechanism, which accurately identifies when a concept drift occurs and orchestrates re-training as a targeted mitigation strategy. This proactive approach allows the system to dynamically adapt to changing data distributions, ensuring that the model remains accurate and relevant over time.

Our solution works by continuously monitoring the performance of the trained models. Based on the performance feedback, it determines whether further training is necessary. During training rounds, the system marks the round as active, indicating that the model is being updated and resources are actively engaged. When the system detects that no further training is required—either because the model has converged or a drift has been successfully mitigated—it transitions the round to an idle state. In the idle state, client devices are able to conserve processing resources, such as power consumption, and the communication costs between the server and clients are significantly reduced. By carefully managing these phases of activity and idleness, DareFL ensures a resource-efficient and adaptive federated learning process, optimizing both client and network infrastructure usage while maintaining high model accuracy.

In our scheme, model training is divided into training rounds, similar to Vanilla FL. Prior to each round the server employs DareFL to determine if the ML model needs further training (active round) or not (idle round). In an active round, the server performs the process of learning similar to Vanilla FL *i.e.*, random selection of K trainers and M testers. Upon round completion, the server receives the trained models from the trainers and the *kpi* values from the testers (see Eq. 3.1). The acquired *kpi* values are used as input for the next round's decision. In an idle round, no additional training occurs; clients do not update their models and only inference and *kpi* collection are carried out. DareFL is activated for the first time in the beginning of the second round, when *kpi* values from the first round are available. In the end of each round r, the server collects a $kpi^{c,r}$ value

from each tester $c \in [1, M]$ and forms a *kpi* list, denoted as $\{kpi\}$. The algorithm includes the drift detection (DD) and the convergence detection (CD), both of which require $\{kpi\}$ as input. If drift is detected or convergence is not reached, DareFL will resume training (active round), otherwise training is skipped (idle round).

Drift detection algorithm

Drift detection (DD) is based on the centralized DDM algorithm [66]. DDM operates without accessing training data or using reference data windows (see Sec. 3.2), which is aligned with the principles of FL. It accepts a sequence of 1s and 0s (0 for a successful classification, 1 for mis-classification). Assuming p_i and s_i is the error rate and standard deviation at the sequence's instance i and p_{min} and s_{min} the minimum recorded values, respectively then:

$$DDM(\beta_2, \beta_3) = \begin{cases} \text{warning}, & \text{if } p_i + s_i \geq \beta_2 \cdot s_{min} \\ \text{drift}, & \text{if } p_i + s_i \geq \beta_3 \cdot s_{min} \\ \text{no drift}, & \text{otherwise} \end{cases} \tag{3.2}$$

We adapt its functionality to allow for distributed drift detection as follows. Each client's *kpi* value is transformed to 0 or 1, by comparing to a tunable parameter β_1 (see Algorithm 1, line 3). β_1 is a threshold that expresses the minimum ML model's accuracy improvement over the baseline (naive) algorithm's accuracy, to account for a successful classification. For example if $\beta_1 = 5\%$, any ML prediction that does not surpass the naive algorithm's accuracy by at least 5% is considered as mis-classification. Its value is tuned during the "warm-up" period's test-runs that provide initial statistics over the ML model's performance (see Sec. 3.3.1). Note that the "warm-up" period is mandatory for any ML scheme during initialization, thus our approach does not introduce any additional cost resource-wise. The DDM is fed with the transformed $\{kpi\}$, which we denote as (DDM list) $\{ddm\}$ so that a potential drift can be detected (see Eq. 3.2) [66]. DDM's sensitivity parameters β_2 and β_3 (see Eq. 3.2) can be tuned via grid-search or via ML techniques *e.g.*, reinforcement learning, which goes beyond this work's scope.

Convergence detection algorithm

For the convergence detection (CD), we calculate the central tendency *ckpi* of each round's *kpi* list $\{kpi\}$, as the average value of all its elements (see Algorithm 2, line 2). The server keeps track of all *ckpi* values (in a per-round basis) in a *ckpi* list, denoted as $\{ckpi\}$ (see Algorithm 2, line 3). Note that *kpi* values provide only an evaluation of the ML model performance in each client and do not reveal private information, fully aligning with the FL principles. Convergence detection is then achieved using the following rule: if *ckpi* is not improved over the last β_4 rounds we assume that convergence is reached (see Algorithm 2, line 4). This rule is an analogy to the early-stopping concept (see Sec. 3.3.1). β_4 expresses the average number of rounds required for an FL convergence and may vary depending on the ML model architecture and the ML task. Tuning is performed during the "warm-up" period, by monitoring each round's accuracy to observe its rate of change.

3.4 Simulation environment

DareFL is designed to operate in any type of FL task and use-case (see Sec. 3.3). For its evaluation we have selected the case of predictive Quality of Service (pQoS) *i.e.*, prediction

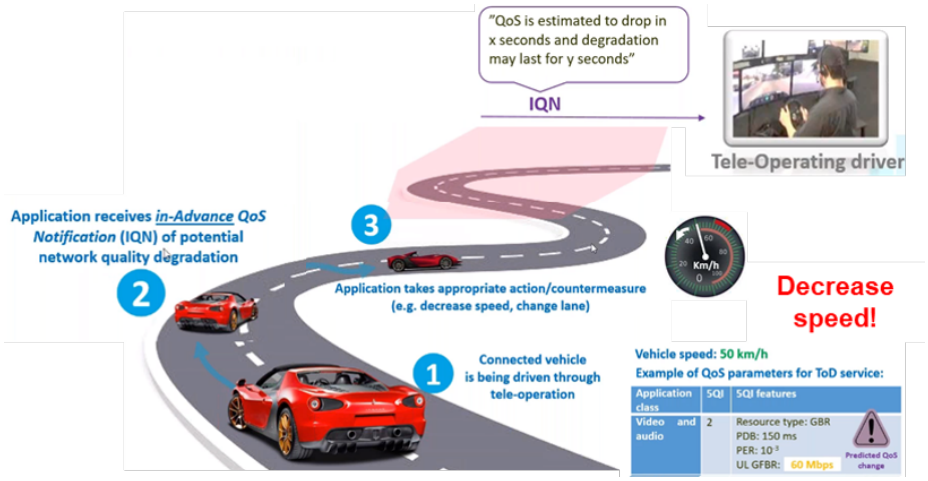


Figure 3.2: The concept of predictive QoS (pQoS) - 5GAA

of network QoS parameters in vehicular environments. In the absence of public pQoS datasets with drift instances, we have fabricated two synthetic datasets[1] via a high fidelity network simulation that realistically capture distinct drift scenarios in dynamic mobile environments. On top, we have built a Python-based distributed ML simulator[2], to utilize the datasets and evaluate our proposal. Both the datasets and the ML simulator source code are publicly available.

3.4.1 The use-case of pQoS

Automotive applications, such as automated driving (AD) functions, tele-operated driving (ToD), platooning, and others, are expected to bring a wide range of benefits, including enhanced road safety, improved traffic efficiency, and increased driving comfort. These applications, however, depend heavily on the connectivity provided by mobile networks, which in turn impose stringent quality of service (QoS) requirements. These requirements may include minimum throughput, maximum delays, and other performance metrics that are crucial for the proper functioning of such applications. Failure to meet these QoS requirements, often due to unpredictable network conditions, can result in service degradation. This degradation not only impacts the user experience but, in critical scenarios, may compromise safety by affecting the timely and accurate functioning of the systems involved.

To mitigate such risks and ensure the reliable performance of automotive applications, the concept of predictive QoS (pQoS) has been introduced, as highlighted by the 5G Automotive Association (5GAA) [70]. pQoS serves as a proactive solution to anticipate and address potential connectivity issues before they impact service quality. It provides the vehicle with advance notifications about potential degradation in cellular connectivity, allowing the system to prepare for or mitigate the effects of these disruptions, as shown in Fig. 3.2. This foresight is particularly valuable for applications requiring real-time data exchange and high reliability.

The pQoS mechanism relies on a variety of features related to both network properties and client mobility to make these predictions. These features may include signal strength, vehicle position, speed, and other mobility-related parameters[71]. By leveraging these

[1] https://zenodo.org/records/11084689
[2] https://github.com/gdrainakis/distributed_pqos

variables, pQoS can forecast critical network QoS parameters, such as throughput, enabling the system to act proactively to prevent service failures. For example, if a potential dip in throughput is predicted due to a vehicle's movement through an area with known coverage issues, the system can adjust communication strategies or make decisions that ensure continuous service without interruptions. This predictive approach helps maintain the high standards required by automotive applications, ensuring that network performance aligns with the stringent needs of safety-critical systems.

pQoS can be provided through different deployment models, primarily by the mobile network operator (MNO) or in an over-the-top (OTT) fashion by third-party entities. When implemented by the MNO, pQoS benefits from the operator's comprehensive view of the network's statistics and performance. This allows for centralized monitoring and prediction of QoS parameters across the entire network. However, another alternative is deploying pQoS in an OTT manner, which typically involves third parties such as car manufacturers [70]. One of the key advantages of the OTT approach is its broader geographical coverage compared to the MNO-based solution, as it is not constrained by the MNO's network boundaries.

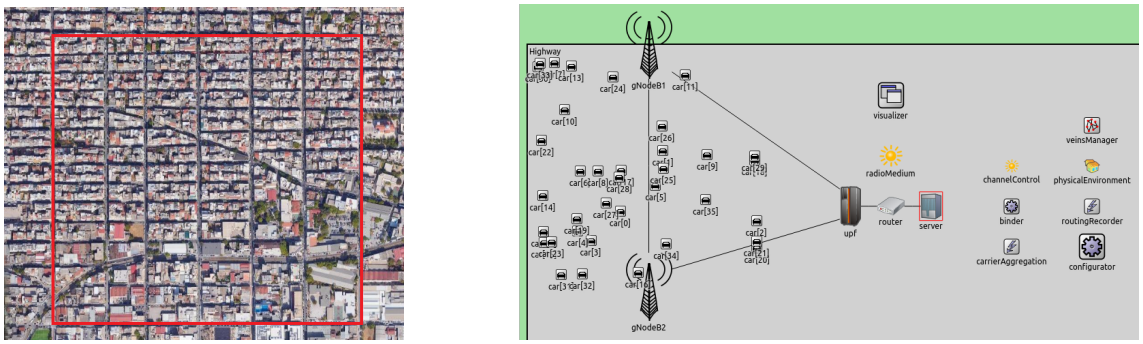
While both the MNO-based and OTT-based approaches have distinct deployment benefits and trade-offs [70], our focus is on the OTT model. The OTT model leverages data gathered directly from client devices, thus overcoming potential limitations related to data confidentiality that might arise in the MNO-based approach. In this model, client devices can collect and share their own data, including mobility parameters and local network statistics, enabling more granular and personalized predictions. By utilizing client-side data, the OTT approach can adapt to rapidly changing environments and provide a level of flexibility that is often not feasible within the confines of the MNO's control.

The insights provided by pQoS can then be used to dynamically adapt automotive applications to maintain optimal performance despite changing network conditions. For example, in ToD scenarios, pQoS can predict network issues and trigger a speed change to ensure that the vehicle's control system operates smoothly despite potential communication delay [84]s. In the case of video streaming, pQoS can predict bandwidth reductions and adjust video resolution accordingly, preventing buffering or service interruptions. These adaptations help ensure that the system meets the stringent real-time requirements of automotive applications, despite fluctuations in network performance.

Overall, pQoS offers high applicability and practicality, particularly in dynamic mobile environments like those encountered in automotive scenarios. Its integration into such systems allows for proactive management of network resources, ensuring that services can continue uninterrupted even in the face of unexpected connectivity challenges. Furthermore, cellular environments are known to experience frequent drifts in network performance for a variety of reasons, including mobility, interference, and congestion [71]. This makes pQoS a particularly well-suited application for drift management, providing a realistic and valuable context for evaluating and refining drift detection and mitigation techniques.

3.4.2 pQoS formulation as an ML task

In our approach pQoS is addressed as a *multi-variate multi-step* times-series problem [81]. The term multi-variate suggests that multiple features (network-related, mobility, *etc.*), are considered as input features (as opposed to uni-variate). The term multi-step expresses the solution's ability to predict several steps ahead in time (prediction horizon). Assuming that the input (raw) data collected by each client is a time-series vector consisting of a



(a) (b)

Figure 3.3: Synthetic dataset generation: (a) import of real world map and (b) network and mobility co-simulation

total of j features (columns) and i samples (rows), where t_i marks the timestamp, x_i^j the independent variables *i.e.*, the network-related and mobility-related input features and y_i the dependent variable *i.e.*, the variable we wish to predict (in our case that is a QoS parameter), the data can be formulated as an ML dataset $\boldsymbol{D}_{i\times j}$ as (bold letters denote a vector):

$$\mathbf{D}_{i\times j} = \begin{bmatrix} t_1 & x_1^1 & .. & x_1^{j-2} & y_1 \\ t_2 & x_2^1 & .. & x_2^{j-2} & y_2 \\ .. & .. & .. & .. & .. \\ t_i & x_i^1 & .. & x_i^{j-2} & y_i \end{bmatrix} \tag{3.3}$$

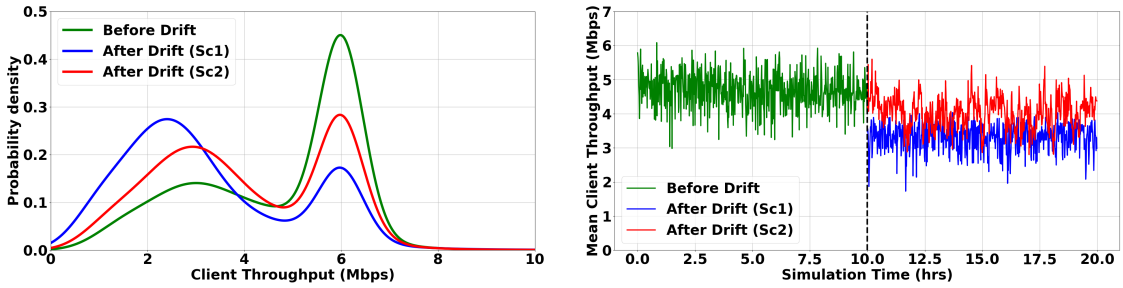
The time-series dataset is then restructured as a supervised ML problem via the sliding window technique that addresses previous time steps as input variables and next time steps as output variables. Assuming a window time w ($w < i$) and denoting the transpose of a vector $\boldsymbol{A}$ by $\boldsymbol{A}^T$, the restructured dataset $\boldsymbol{D}'$ becomes:

$$\begin{aligned} &\boldsymbol{D}'_{(i-w)\times(j\cdot w+1)} = [\boldsymbol{d}_1, \boldsymbol{d}_2, ..., \boldsymbol{d}_k, ..., \boldsymbol{d}_{i-w}]^T, where: \\ &\boldsymbol{d}_k = [x_{k-w}^1, .., x_{k-1}^1, .., x_{k-w}^{j-2}, .., x_{k-1}^{j-2}, .., y_{k-w}, .., y_{k-1}, y_k] \end{aligned} \tag{3.4}$$

The computation of the baseline naive predictor, serving as input to Convergence (CD) and Drift Detection (DD) routines of DareFL (see Algorithm 1 and 2) is carried out by a rolling-means algorithm (relevant to time-series tasks); assuming a time-series in the form of Eq. 3.3, the (naive) prediction $\widehat{y}_i$ of the sample's i dependent variable y_i (ground truth) in time t_i is the mean value of the dependent variable's last w values, where w stands for the time-window of Eq. 3.4: $\widehat{y}_i = \frac{1}{w}\sum_{x=i-w}^{i-1} y_x$ with $i < w$. For the respective accuracy metric used in DareFL to compare the naive with the ML prediction (see *kpi* calculation in Eq. 3.1) we have selected RMSE, as the most relevant to time-series tasks [81].

3.4.3 Generating pQoS datasets with concept drift

Our synthetic datasets represent a dynamic environment, where several clients (vehicles) are moving in an urban area. Each client runs a streaming cloud service constantly receiving User Datagram Protocol (UDP) data packets. This data can refer to various automotive applications *e.g.*, High-Definition digital maps for AD, audio commands for ToD, video for infotainment services, *etc.* The QoS parameters *e.g.*, throughput of each



(a) (b)

Figure 3.4: Effects of drifts in client throughput: (a) probability density function and (b) average throughput

client's service change in the course of time, depending on the vehicle position, the network's physical layer properties, *etc.* Network simulation is performed using Simu5G, a library that emulates a 5G cellular environment in OMNeT++ [85]. The simulator's radio parameters *e.g.*, channel properties, antenna settings, *etc.*, are set according to the Macro-cell model proposed by International Telecommunication Union [86]. The map in our simulations comprises of an urban $600 \times 600\ m^2$ area located in a suburb of a European capital (see Fig. 3.3a).

Inside this area four 5G base-stations (gNodeBs) have been installed by the national network operator enabling four 5G cells [87]. This area, divided into several blocks by the actual road network is integrated in our simulation by an OpenStreetMap (OSM) instance [88]. The total number of included vehicles is set to 25, according to vehicle density statistics in the corresponding country [89]. The road network's traffic is simulated by SUMO, a traffic simulation package [90] that creates a digitized version of the (real-world) OSM map and produces the route files for the vehicles. Route files are loaded in the Simu5G simulator, where a network-vehicular mobility co-simulation takes place (see Fig. 3.3b). For each vehicle's route we assume SUMO's default parameters for urban environment *i.e.*, exponential speed model (with maximum speed restriction as defined by the OSM traffic rules) and the probability matrix at intersections for {lane keeping, turn left and right} as {0.5, 0.25 and 0.25}, respectively. The following information is collected for each vehicle using OMNeT++'s monitoring service: *timestamp*, *channel quality indicator*, *packet delay*, *measured signal to noise ratio (SNR)*, *client position (x,y,z)*, *client velocity (x,y,z)*, *received SNR*, *radio link control throughput*, *serving cell*, *client throughput*. These features are sampled at 1 Hz and comprise the values of our synthetic time-series QoS dataset.

For the considered use-case we have created two drift datasets that correspond to complementary cases of major long-term changes in the considered environment: 1) a network infrastructure-driven scenario (Sc1) and 2) a human behavior-driven scenario (Sc2). In Sc1 we assume that two out of four gNodeBs are switched off under a cost-reduction on/off policy to address electricity costs [91] or an infrastructure-share strategy (adopted by MNOs) [92],[93] that would imply such changes. For Sc2 we consider drifts related to the modification of the users' mobility pattern. We assume that a "hotspot" *e.g.*, a metro station is created in the lower-right edge of the map resulting in a traffic increase to that area [94]. This is achieved by increasing the probabilities of the routes leading to the "hotspot" in SUMO's route planning. All generated datasets have a total duration of 20 hrs (simulation time) and the respective drift instance is introduced at $t = 10$ hrs.

The effect of each drift instance on the clients throughput pattern is depicted Fig. 3.4a and Fig. 3.4b. Prior to drift, clients achieve an average throughput of 4.7±1.65 Mbps. This is decreased to 3.32±1.79 and 4.03±1.74 Mbps for Sc1 and Sc2, respectively. These changes on the underlying data distribution will eventually be reflected on the prediction's model accuracy, as we will demonstrate in Sec. 3.5.

3.4.4 Distributed ML simulator

Our simulator implements the FL framework of Sec. 3.3, along with the involved client-server communication and ML processing (consumption) costs. The distributed ML (training and inference) is facilitated via Pytorch, a Python deep ML library [80]. For the network resource consumption modelling, we assume a setup with a centralized cloud server and several client-vehicles. Each vehicle, equipped with a 5G modem, communicates with the cloud server via the 5G cellular network. For the processing tasks (training, inference) the server is assumed equipped with a Graphics Processing Units (GPU), while vehicles avail less powerful processing capabilities utilizing a common CPU [95]. By loading the pQoS datasets to the simulator, each client's features *e.g.*, position, speed, current throughput, *etc.*, are obtained by querying the respective client dataset.

The developed simulator also estimates the energy consumption imposed to the clients and the server due to data processing and transmission. To do so, we rely on values taken from credible measurements in literature and device bench-marking: A typical 5G modem at uplink and downlink data rates of 20 and 100 Mbps respectively, consumes for transmission and reception 2.5 and 3.5 Watt respectively [96]. A typical CPU trains an ML model relevant to pQoS data (see Sec. 3.4.2) at a speed of 25 samples/sec, while a GPU reaches 900 samples/sec [97]. Those tasks require power of 200 Watt [98] and 225 Watt [97], respectively. Model aggregation on the other hand, has not been measured in literature thus we estimate the server's GPU computational speed and consumption based on the computationally-similar matrix-to-matrix multiplication. As in [99], the consumption cost is set to 100 Watt and the aggregation speed averages to 10 models/sec; with both values also validated by our processors readings.

3.5 Simulation-based evaluation

3.5.1 Evaluation methodology

In our results we perform a three-step evaluation. Firstly, we compare distributed pQoS against the typical centralized solution (CL) and explore the involved performance and resource-consumption trade-offs. For a fair CL-FL comparison: 1) we assume no drift occurrence, 2) contrary to Sec. 3.2 works, we use *both* synthetic and real data for cross-validation of our results. Our evaluation focuses on pQoS performance *i.e., prediction of client throughput* rather than any involved adaptation actions (in case of QoS degradation), which are application-dependant. We run experiments using a) our first synthetic pQoS dataset (Sc1) prior to drift (see Sec. 3.4.3) and b) BerlinV2X, a public pQoS dataset from a measurement campaign of 4 vehicles driving in Berlin on highway and urban routes[100]. Each run is repeated 10 times so that the selected vehicle-clients are changed, for a total of 40 experiments (2 algorithms × 10 repetitions × 2 datasets).

Secondly, we study distributed pQoS under drift; we compare our novel drift management FL algorithm (DareFL) against existing solutions: 1) *Vanilla FL* (see Sec. 3.3.1), 2) *Continuous FL* (ConFL - see Sec. 3.2) and 3) *Adaptive FL (AdaptFL)* [69] under the

two pQoS drift scenarios (Sc1, Sc2) described in Sec. 3.4.3. Vanilla and ConFL serve as baselines, while AdaptFL is selected as a representative SotA FL algorithm that manages drift without introducing any additional complexity in the clients-side (see Sec. 3.2). Each scenario is repeated 10 times for a total of 80 experiments (2 scenarios $\times$ 4 algorithms $\times$ 10 repetitions). QoS (throughput) prediction accuracy across time is evaluated via the following metrics, typically used for prediction error (mean values and standard deviations across all clients): *Root Mean Square Error (RMSE)*, *Mean Absolute Error (MAE)* and *Symmetric Mean Absolute Percentage Error (SMAPE)* [74], [81]. For the respective resource consumption: 1) *Normalized communication cost i.e.,* total data exchanged between the server and the clients normalized to the ML model size and 2) *Clients* and *Cloud energy cost i.e.,* the total energy consumed at each side for processing and transmission (see Sec. 3.4.4).

Finally, we provide an extensive comparative analysis of DareFL's drift detection capability against AdaptFL. We repeat each scenario 100 times and record the drift detection outcome in every round, for a total of 24000 samples (2 scenarios $\times$ 2 algorithms $\times$ 100 repetitions $\times$ 60 rounds). In both scenarios drifts occur at half-time, which constitutes the experiment's ground truth in terms of drift occurrence. Drift detection is then evaluated as a binary classification problem (True/False) via *Accuracy*, *Precision*, *Recall*, *Specificity* and *F1 Score*.

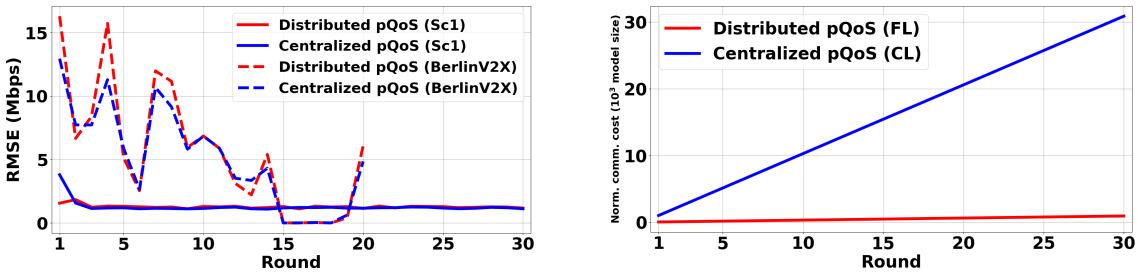
Throughout the experiments, round duration is set to q=1200 secs *i.e.,* a total of R=60 rounds for Sc1, Sc2 and R=20 for BerlinV2X datasets. In Sc1 and Sc2, drift occurs at half-time (R=30) and lasts throughout the experiment. Drift effects on the models need a maximum of 7 rounds to take place, as measured during the "warm-up" phase (maximum rounds DareFL required for convergence after drift). For Sc1, Sc2 the total number of clients is set to 25, with K=5 trainers and M=20 testers per round, based on [101]. For BerlinV2X (total of 4 clients) we set K=M=2. Trainers' data is split at a typical 80%-20% ratio [81]. Prior to all simulations, all ML models are tuned (tuning statistics are omitted for clarity) during the "warm-up" phase (see Sec. 3.4.2).

Training is performed using an LSTM model (an established predictor for sequential data [102]), consisting of: 11 input features (equal to the total client-acquired features of each dataset) and 8 output features *i.e.,* a (throughput) prediction horizon of 8 sec (other automotive ML-based predictions show acceptable accuracy up to a 5 sec horizon [103]). Note that throughout Sec. 3.5.2 we show the results for a horizon of 6 sec for clarity, though the same principles apply to all other horizons up to 8 sec. For the LSTM we set: sliding window w=75, hidden size=50, Min-Max normalization, decay=10^{-5}, Rectified Linear Unit activation and Mean Square Error loss function, based on test-runs and related works on LSTM models[81], [80]. Hyper-parameter tuning on our LSTM model via grid-search resulted in the following values: batch size=64, learning rate=10^{-5}, epochs=500. Leveraging on our test statistics during "warm-up" we set DareFL's parameters: β_1=0, β_4=5 rounds and default values β_2=2 and β_3=3. For fairness, AdaptFL's parameters are also tuned via grid search: β_1=β_2=β_3=0.7.

3.5.2 Evaluation results

Centralized (CL) vs Distributed (FL) pQoS

As portrayed in Fig. 3.5a, distributed pQoS (FL) converges similarly to its centralized alternative (CL), whilst preserving data privacy. This is validated both for the public (BerlinV2X) and our synthetic (Sc1) pQoS dataset. In terms of (throughput prediction) accuracy (averaged across all rounds), CL achieves a maximum improvement of 9% across



(a) (b)

Figure 3.5: CL *vs.* FL pQoS: (a) RMSE and (b) Communication cost comparison

all metrics and rounds over FL for Sc1 and 11% for BerlinV2X (see Table 3.1). Interestingly, both CL's and FL's accuracy in BerlinV2X exhibits up to 2 orders of magnitude higher standard deviation compared to Sc1 across all accuracy metrics (see Table 3.1), as a result of BerlinV2X's limitations *e.g.,* small number of clients. In terms of resource consumption , FL achieves an up to 3100% reduction of communication costs (see Fig. 3.5b), since LSTM models exchanged in FL are typically lightweight compared to training data that is uploaded to the server in CL. Offloading a pQoS task to the clients (FL), leads to a reduction of cloud energy costs by a factor of 3.2×10^5 at $R = 30$ (see Fig. 3.6). The energy costs are distributed to the client-devices, thus the aggregated client energy consumption is increased by a factor of 4.4×10^6 at $R = 30$ (see Fig. 3.6).

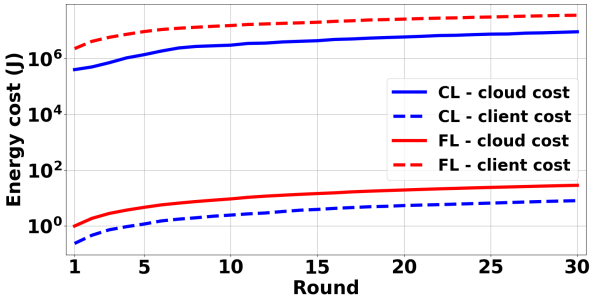


Figure 3.6: CL *vs.* FL pQoS: Energy cost comparison

Table 3.1: CL vs. FL pQoS accuracy (horizon=6 sec)

ML algorithm	RMSE (Mbps)	MAE (Mbps)	SMAPE (%)
CL (BerlinV2X)	4.76±3.54	3.55±2.73	11.55±8.27
FL (BerlinV2X)	5.13±4.35	4.00±3.52	12.45±10.54
CL (Sc1)	1.19±0.08	0.91±0.05	10.93±0.75
FL (Sc1)	1.27±0.12	0.99±0.06	11.80±0.65

DareFL vs. existing algorithms

DareFL's accuracy comparison against existing FL algorithms for Sc1 is shown in Fig. 3.7a. Vanilla FL suffers from a sudden increase (51%) of the prediction error (RMSE) at the 31^{th} round, as a result of the inflicted drift. This is also portrayed in all accuracy metrics (RMSE, MAE, SMAPE), shown in Tables 3.2 and 3.3. In Vanilla FL, the ML model is trained for a limited number of rounds until convergence is reached, thus it cannot adapt to

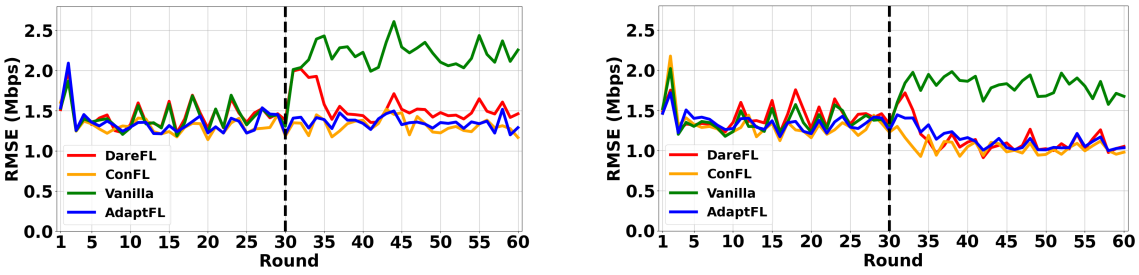


Figure 3.7: RMSE comparison for: (a) Sc1 and (b) Sc2

future drifts; as a result, the model's performance is degraded. ConFL on the other hand addresses drifts by constantly training (updating) the model, thus achieving maximum accuracy at all times. Compared to Vanilla, ConFL exhibits an average of 7% higher accuracy (in terms of RMSE) before drift, 33% during and 40% after drift (see Tables 3.2-3.4). Constant training however, results in the linear increase of the network bandwidth (see Fig. 3.8), energy costs for the clients (see Fig. 3.9a) and the server (see Fig. 3.9b). ConFL consumes almost 300% more bandwidth and 250% more energy (both in the client and in the server-side) compared to Vanilla FL up until the 30^{th} round. The respective values at the end of the simulation are 720%, 470% and 580%. Unlike these solutions, DareFL leverages on its drift detection mechanism to ensure high accuracy, comparable to ConFL. Prior to drift, ConFL (which exhibits the top performance) outperforms DareFL by a maximum of 8% across all accuracy metrics (see Table 3.2).

Table 3.2: Comparison of pQoS FL algorithms accuracy under drift, before drift (Round 1-30) (horizon=6 sec)

FL algorithm	Before drift (Round 1-30)		
	RMSE (Mbps)	MAE (Mbps)	SMAPE (%)
Vanilla (Sc1)	1.42±0.16	1.16±0.15	13.32±1.20
DareFL (Sc1)	1.43±0.16	1.15±0.16	13.30±1.28
AdaptFL (Sc1)	1.36±0.17	1.11±0.17	12.94±1.19
ConFL (Sc1)	**1.32±0.14**	**1.06±0.14**	**12.50±1.21**
Vanilla (Sc2)	1.40±0.17	1.12±0.16	12.95±1.27
DareFL (Sc2)	1.42±0.15	1.11±0.14	12.96±1.38
AdaptFL (Sc2)	1.35±0.11	1.09±0.10	12.83±0.79
ConFL (Sc2)	**1.32±0.19**	**1.05±0.18**	**12.34±1.47**

Table 3.3: Comparison of pQoS FL algorithms accuracy under drift, during drift (Round 31-37) (horizon=6 sec)

FL algorithm	During drift (Round 31-37)		
	RMSE (Mbps)	MAE (Mbps)	SMAPE (%)
Vanilla (Sc1)	1.99±0.38	1.76±0.30	23.13±4.10
DareFL (Sc1)	1.71±0.26	1.44±0.25	20.31±3.17
AdaptFL (Sc1)	1.36±0.08	1.08±0.08	16.60±1.94
ConFL (Sc1)	**1.32±0.11**	**1.00±0.09**	**15.39±1.72**
Vanilla (Sc2)	1.70±0.24	1.49±0.21	18.20±3.11
DareFL (Sc2)	1.36±0.24	1.05±0.23	13.14±1.82
AdaptFL (Sc2)	1.32±0.10	1.05±0.09	13.76±0.97
ConFL (Sc2)	**1.14±0.15**	**0.86±0.11**	**11.24±0.98**

Upon drift occurrence, DareFL adapts in a handful of rounds (an average of 5.3 rounds

Table 3.4: Comparison of pQoS FL algorithms accuracy under drift, after drift (Round 38-60) (horizon=6 sec)

FL algorithm	After drift (Round 38-60)		
	RMSE (Mbps)	MAE (Mbps)	SMAPE (%)
Vanilla (Sc1)	2.22±0.14	1.87±0.13	25.08±1.15
DareFL (Sc1)	1.48±0.08	1.19±0.08	18.37±0.98
AdaptFL (Sc1)	1.35±0.08	1.07±0.07	17.21±0.99
ConFL (Sc1)	**1.32±0.09**	**1.00±0.08**	**15.89±1.09**
Vanilla (Sc2)	1.80±0.11	1.54±0.09	19.52±1.27
DareFL (Sc2)	1.07±0.08	0.84±0.07	11.50±0.67
AdaptFL (Sc2)	1.08±0.07	0.83±0.06	11.76±0.77
ConFL (Sc2)	**1.02±0.06**	**0.78±0.06**	**10.77±0.70**

across all experiments) and sustains similar accuracy to that of ConFL until the end of the experiment (see Fig. 3.7a). Specifically, ConFL outperforms DareFL by a maximum of 16% across all accuracy metrics after drift (see Table 3.4). Meanwhile, DareFL's energy and bandwidth footprint is kept relatively low (comparable to Vanilla FL), grace to its convergence detection mechanism that facilitates idle training rounds *i.e.,* saving on resources. As a result, DareFL achieves 76% lower communication costs (see Fig. 3.8) and 68% lower energy costs in the clients (see Fig. 3.9a) and 74% on the server (see Fig. 3.9b), compared to ConFL at the end of the simulation.

AdaptFL, which serves as a SotA drift management FL algorithm has similar behavior to ConFL, since it assumes constant training. As such, its resulted accuracy is comparable to that of ConFL (see Fig. 3.7a). Note that AdaptFL achieves the second best performance in all accuracy metrics (see Tables 3.2-3.4); AdaptFL outperforms DareFL by a maximum of 10% across all metrics before and after drift and 25% during drift, due to constant training. However, it exhibits the highest resource consumption compared to all other algorithms. Specifically, its communication costs are equal to that of ConFL (see Fig. 3.8 - ConFL and AdaptFL lines coincide) *i.e.,* 720% more than Vanilla and 320% more than DareFL. Interestingly, AdaptFL consumes 200% more energy compared to DareFL and 100% more than ConFL in the clients and in the server side by the end of the simulation. This behavior occurs due to AdaptFL's drift management mechanism; at the server side it requires additional calculations for its detection process, similar to FedAvg's aggregation process (see Sec. 3.2). The gradual adaptation of the learning rate also leads to "slower" learning in the clients-side *i.e.,* additional epochs, which increases the client energy footprint. These excessive energy costs however, have no effect on the increase of accuracy.

The findings of Sc1 are also validated in Sc2 (see accuracy comparison in Fig. 3.7b). Vanilla FL experiences a 43% accuracy drop (RMSE is increased) at the 31^{th} round, which marks the effect of Sc2's drift on the model's performance. ConFL exhibits the best performance across time, due to constant training. ConFL outperforms Vanilla across all metrics by at least 5%, 32% and 43% before, during and after drift respectively (see Tables 3.3, 3.4). DareFL exhibits similar performance to that of Vanilla FL prior to drift however, it adapts after the the 31^{th} round, due to its drift detection mechanism. As such, it converges in similar accuracy levels as ConFL; ConFL achieves a maximum of 7% accuracy improvement across all metrics after drift *vs.* DareFL (see Table 3.4). Compared to DareFL, AdaptFL achieves no more than 4% accuracy improvement before drift and 2% after drift. Similarly to Sc1, both AdaptFL and ConFL consume multiple times more energy and bandwidth to achieve these improvements over DareFL. Note that the induced cost values for Sc2 are omitted, since the behavior is identical to that of Sc1 (see Fig. 3.8,

3.9a and 3.9b).

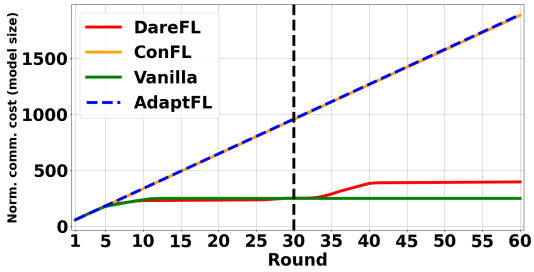
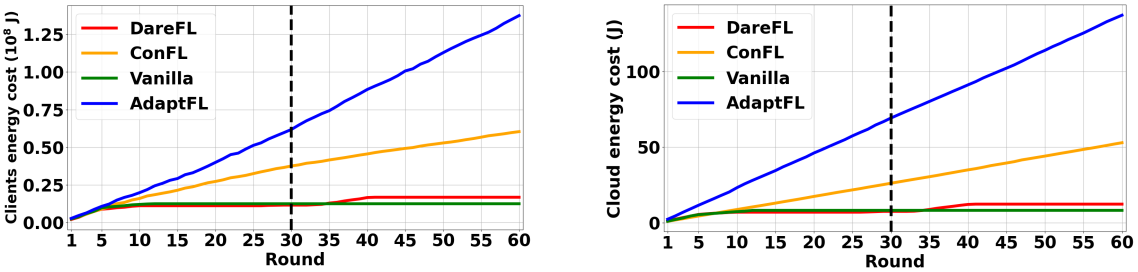


Figure 3.8: Sc1 - Communication costs



(a) (b)

Figure 3.9: Energy consumption for Sc1 in the (a) clients and (b) the cloud side.

DareFL drift detection capability

The drift detection comparison between DareFL and AdaptFL for Sc1 and Sc2, formulated as a binary classification problem is presented in Fig. 3.10. DareFL demonstrates a 6% higher detection accuracy (ratio of successful to total detections) for both scenarios. Since the samples under comparison (drift/no drift) are not balanced in the two concerned scenarios (drift instances account for almost 10% of the total instances), the accuracy metric stands as a high-level spot check for the two algorithms' performance. Interestingly, DareFL exhibits higher precision *i.e.,* the ratio of true positives to the predicted positives (by 21% for Sc1 and by 26% for Sc2) and specificity *i.e.,* the ratio of true negatives to all negative outcomes (by 8% for Sc1 and by 7% for Sc2) compared to AdaptFL. However, DareFL lacks in terms of recall *i.e.,* the ratio of true positives to the actual positives (by 14% for Sc1 and by 8% for Sc2).

This behavior suggests that DareFL minimizes false positives *i.e.,* the (false) detection of drift event when no drift has occurred, while AdaptFL minimizes false negatives *i.e.,* no detection of drift during a drift instance. In a sense, AdaptFL serves as a more sensitive drift detection algorithm compared to DareFL, which is more drift-tolerant. For pQoS and the drifts under consideration these metrics are of equal importance; false positives lead to unnecessary training and therefore resource waste, while false negatives may degrade the prediction performance of the ML model. The algorithms' overall detection performance is better assessed by the F1 score metric *i.e.,* the harmonic mean of precision and recall, where DareFL outperforms AdaptFL (3% for Sc1 and 9% for Sc2). It is worth noting that solely for drift detection, AdaptFL consumes up to 3 times more energy in the server compared to DareFL. This is due to the fact that AdaptFL requires more complex calculations that involve the parameters of the received client models (see Sec. 3.2), as

opposed to DareFL that only calculates the average of a list of metrics, received from the clients (see Sec. 3.3.2).

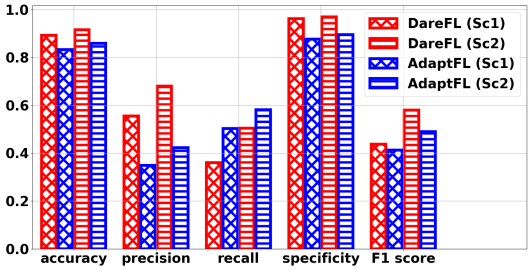


Figure 3.10: Drift detection comparison

3.6 Conclusion

In this chapter, we have explored the emerging paradigm of Lifelong Machine Learning for distributed environments, with a specific focus on FL settings. Our research is primarily concerned with the phenomenon of concept drift, which refers to the changes in the statistical properties of client data distributions over time. This drift is a major factor contributing to the degradation of ML models, particularly in dynamic and evolving environments such as mobile networks and automotive systems. As these environments are subject to constant changes, the challenge of maintaining model performance amidst drift becomes particularly significant.

To address this challenge, we have introduced DareFL, a novel and efficient algorithm designed to manage concept drift within the context of FL. DareFL is built on the core principles of FL, such as data privacy and decentralization, and is specifically designed to minimize resource consumption. This is achieved without compromising the ability to detect and mitigate drift, offering a significant improvement over existing SotA techniques. Unlike many prior studies, which often focus on tasks with limited applicability—such as digit recognition or static datasets—our work takes a broader approach. We specifically investigate the case of predictive QoS in dynamic automotive environments, where concept drift is more pronounced and driven by a wider variety of factors, such as mobility, network conditions, and environmental changes.

To overcome the challenge of a lack of publicly available QoS datasets that include instances of drift, we have developed two complementary QoS drift scenarios that are based on both infrastructure- and user-related factors. These scenarios are generated using our open-source, high-fidelity network and mobility simulator. This simulator, which models both network and vehicle mobility in great detail, provides a realistic testing environment for the evaluation of FL algorithms. By leveraging this simulator, we are able to assess the performance of DareFL under the influence of drift, comparing it against a set of baseline algorithms. These include: a) the Vanilla FL algorithm, as presented by Yin et al. [[72]], b) a continuous training approach, and c) a state-of-the-art drift mitigation solution, as proposed by Canonaco et al. [[69]].

Our simulation results across the two distinct drift scenarios reveal that DareFL outperforms the baseline approaches in terms of resource efficiency. Specifically, DareFL reduces the overall resource consumption by up to 70%, including savings on network infrastructure, client devices, and the central server. This significant reduction in resource usage comes at a modest cost in terms of accuracy, with DareFL exhibiting an average

accuracy drop of only 10% compared to the more resource-intensive competing schemes. This demonstrates that DareFL strikes an effective balance between resource efficiency and model performance, making it particularly suitable for deployment in resource-constrained environments such as mobile networks and automotive systems.

Looking ahead, several promising directions for future research emerge from this work. First, the proposed framework could be evaluated and validated in a real-world 5G testbed, where actual network conditions and mobility patterns could provide further insights into the algorithm's practical applicability. Second, the current results could be generalized to a broader set of drift scenarios, considering additional environmental factors or other types of concept drift beyond the ones studied here. Finally, a potential avenue for future improvement is the optimization of DareFL's parameters through the use of ML techniques, such as reinforcement learning, to adapt the algorithm's behavior to different environments and use cases more effectively.

4

Practical implementation

4.1 Introduction

In the previous chapter, we examined the dynamics of FL over time, specifically focusing on how changes in the underlying client data distribution, commonly known as concept drift, impact the performance of FL models during both training and inference. We proposed an end-to-end framework designed to detect and mitigate these effects in a resource-efficient manner, with a particular emphasis on volatile environments prone to concept drift, such as vehicular and mobile networks. To illustrate the practicality of our solution, we introduced the automotive use case of QoS prediction and demonstrated its effectiveness through various simulation scenarios. This approach enables the deployment of FL systems in large-scale, real-world settings that can adapt dynamically to environmental changes.

In the present chapter, we extend our theoretical findings by investigating their application in practical scenarios involving real-world commercial networks, IoT devices capable of performing FL, and relevant applications. The primary focus of this chapter is the practical implementation of DML schemes within the automotive sector, particularly in the context of CCAM applications.

The structure of the chapter is as follows. Sec. 4.2 provides a detailed description of our measurement campaign, during which we collected a real network QoS dataset. This dataset serves as the foundation for training AI/ML models for pQoS, as discussed in Sec. 4.3. In Sec. 4.4, we describe the deployment of a distributed pQoS service on actual vehicles, which act as Extreme-Edge/IoT devices, and present the valuable experimental results obtained from this deployment in Sec. 4.5. Finally, in Sec. 4.6, we summarize the chapter and outline future research directions.

4.2 pQoS measurements

4.2.1 Motivation

CCAM applications are expected to transform the mobility sector, enabling safer, more efficient, and sustainable transportation systems. These applications encompass a wide range of functionalities, including tele-operated driving, infrastructure-assisted environmental perception, cooperative lane merging, and advanced features such as 5G-enabled cross-border corridors and platooning initiatives [104], [105]. Central to the success of these applications is reliable mobile network connectivity, as many CCAM use cases depend on stringent QoS guarantees such as ubiquitous network coverage, minimum data rates, and low-latency communication [70].

Despite advances in modern cellular technologies, including 5G, that aim to deliver such QoS guarantees, real-world connectivity remains vulnerable to numerous environmental and technical factors [70]. These variables can severely impact achievable QoS, posing significant risks to user experience and, more critically, the safety of automated systems [104].

To address these challenges, the concept of pQoS has been introduced [70]. This approach estimates future QoS values and proactively informs automotive applications about potential QoS degradation events. Such proactive mechanisms allow automotive systems to adapt their functionality in response to predicted QoS changes, enabling operations such as speed reduction, fail-safe maneuvers, or the abortion or completion of specific operations [70].

Estimating QoS in vehicular environments is a particularly complex task, given the rapid temporal variations in radio conditions [71]. As ML techniques increasingly underpin pQoS systems, there is an ever-growing demand for high-quality QoS datasets to train these models effectively. However, the data acquisition process faces several challenges, such as the high cost of measurement campaigns and variability in data quality [71].

QoS values are influenced by a range of factors, including network and radio parameters, user mobility patterns, and spatio-temporal effects [71]. Consequently, careful design of data collection processes is essential to ensure diverse feature representation across various mobility and network scenarios [71].

Although numerous measurement campaigns have been conducted [106], [107], [108], existing QoS datasets present several limitations that restrict their applicability for pQoS tasks. Key shortcomings include: a) A predominant focus on LTE radio-access network (RAN) technologies, with limited coverage of 5G and beyond, b) Measurements concentrated in urban or indoor locations, neglecting the impact of high-speed mobility environments such as highways, which are critical for vehicular applications and c) A lack of cross-border measurements, disregarding the significant effects of roaming on QoS, a major challenge for CCAM systems [109].

To address these limitations, we present *NordicDat*, a novel QoS dataset obtained during a measurement campaign spanning three European countries: Finland, Sweden, and Norway. NordicDat includes 25 hours of driving data featuring diverse speed profiles—characteristic of highway driving—and captures both physical and network-layer features alongside vehicle kinematics. Measurements were conducted near national borders to specifically capture the impact of roaming on QoS. Notably, our dataset includes traces from both LTE and 5G RAN technologies.

Our analysis reveals that changes in roaming, speed profiles, and RAN technologies significantly influence QoS values. NordicDat is leveraged to demonstrate distributed pQoS using FL for throughput and delay prediction tasks. To the best of our knowledge, this constitutes the first attempt to explore distributed pQoS using real-world public data. The NordicDat dataset, accompanied by detailed documentation, is publicly available in an open repository [110], providing an invaluable resource for future research on pQoS and addressing critical gaps in the field.

4.2.2 Related work

Training and evaluation of AI/ML-based QoS prediction algorithms frequently rely on datasets derived from network simulations [111, 112]. While such datasets offer controlled environments for experimentation, they are often criticized for their limited ability to replicate the intricate and dynamic patterns observed in real-world network settings [71].

This limitation underscores the growing need for real-world QoS datasets that can comprehensively capture the interplay between network operations, the radio environment, and the behavior of UE.

Over the years, a variety of real-world QoS datasets have been generated through measurement campaigns covering diverse mobility scenarios and RAN technologies (see Table 4.1). Several of these studies focus on mobility contexts relevant to pQoS in automotive applications. However, their utility is often constrained by specific technological and environmental scopes. For instance, many datasets emphasize LTE as the primary RAN technology [113, 114, 106, 108, 115], while neglecting advancements in 5G. Similarly, mobility patterns in these studies are often limited to urban and suburban driving profiles [116, 117], leaving out high-speed scenarios critical for highway environments.

A limited number of studies extend their focus to include 5G networks under highway mobility conditions [118, 96, 119], but these campaigns are generally confined to single-country setups, thereby failing to account for the effects of cross-border roaming on QoS. Meanwhile, another body of research investigates network characteristics under low-mobility scenarios, such as pedestrian movement or stationary conditions [120, 107, 121, 122]. Although a few of these studies explore the impact of roaming across European countries [109, 123, 124], their findings are primarily based on stationary or near-stationary environments, limiting their applicability to pQoS in vehicular contexts.

In addition to outdoor studies, certain datasets have been developed for indoor environments, often leveraging mobility scenarios involving automated guided vehicles (AGVs)[125, 126] or stationary setups such as office environments[127]. These datasets are predominantly tailored for industrial applications, focusing on private 5G networks, device-to-device (D2D) communications, and similar use cases, which differ significantly from the requirements of automotive pQoS research.

In contrast to these existing datasets, our shared QoS dataset addresses critical gaps and fosters advancements in the field of pQoS research. Specifically:

- **Mobility diversity:** The dataset is exclusively developed through drive tests, encompassing various real-world mobility scenarios that are directly relevant to vehicular applications.
- **Technological breadth:** It includes data from both LTE and 5G RAN technologies, providing a comprehensive perspective on contemporary and future wireless network capabilities.
- **Cross-border coverage:** The dataset is uniquely collected in a cross-border area spanning three countries, enabling the study of roaming effects on QoS—an essential but often overlooked aspect of CCAM applications.

By addressing these dimensions, our dataset not only enhances the realism of QoS prediction tasks but also supports broader investigations into mobility-aware network adaptation, making it a valuable resource for advancing pQoS in dynamic vehicular environments.

4.2.3 Measurement setup and data collection

The NordicDat dataset combines measurements from a 5G modem, external positioning sensors and the vehicle internal data from the Controller Area Network (CAN) protocol [128]. The aim was to collect measurement sequences which combine connectivity, positioning and kinematic data of the vehicle in geographical areas which present QoS

Table 4.1: Comparison of public cellular QoS datasets

Dataset	Mobility	RAN	Location	Area Types			
				Urban	Suburban	Highway	Indoor
5G Connected Mobility [113]	driving	LTE	Nuremberg, Germany	✓	✓	✓	
5G Meas [118]	driving, walking, static	5G	Indianapolis and Chicago, US	✓	✓	✓	
5G Wild [96]	driving, walking, static	5G	2 US cities	✓		✓	
5Gophers [116]	driving, walking	5G	Minneapolis, Chicago, Atlanta, US	✓	✓		✓
BerlinV2X [114]	driving	LTE	Berlin, Germany	✓	✓	✓	
Beyond Throughput [106]	driving, static	LTE	Ireland	✓	✓	✓	✓
Lumos5G [117]	driving, walking	mmWave 5G	Minneapolis, US	✓			✓
Rome [119]	driving, walking	LTE, NB-IoT, 5G	Rome, Italy	✓			✓
SRFG [108]	driving	LTE	Salzburg, Austria			✓	
Terminal [115]	driving	LTE	Single city	✓			
5G Beams [120]	walking	mmWave 5G	Chicago, US	✓	✓		
5G Consumption [107]	walking	5G	Campus		✓		✓
5G PHY Latency [121]	walking, static	5G	Minneapolis, US	✓			
Experience [109]	mobile, static	LTE	Italy, Norway, Spain, Sweden, UK, Germany	✓			
MONROE [123]	mobile, static	LTE	Italy, Norway, Spain, Sweden, UK, Germany	✓			
Roaming [124]	static	5G	France, Italy, Spain	✓			
UE Network Traffic [122]	static, emulated driving	LTE	Volos, Greece	✓			
AGV [125]	mobile	D2D	Industrial				✓
IV2V/IV2I+ [126]	mobile	LTE	Industrial				✓
Urban Office [127]	static	LTE	Vienna, Austria				✓

Figure 4.1: Martti research vehicle

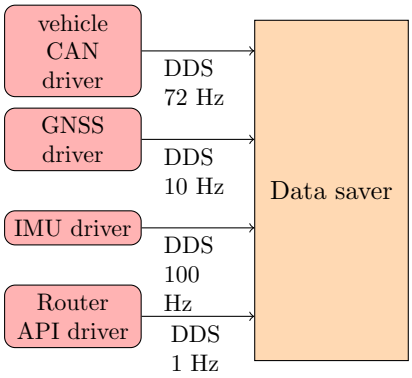


Figure 4.2: Software architecture

degradation. The dataset contains 25 hours of such sequences, collected in arctic rural regions of Finland, Norway and Sweden. The sequences contain sections of low connectivity and total loss of connection, including handover events at national borders.

The dataset was collected utilizing the research vehicle "Martti" (depicted in Fig. 4.1). This vehicle, a modified Volkswagen Touareg, is equipped with a range of external sensors and custom installations designed for advanced self-driving research purposes. The dataset encompasses a comprehensive set of recorded features, all of which are detailed in Table 4.2. These features are categorized as follows:

- **Physical-layer parameters:** This category includes key metrics such as Reference Signal Received Quality (RSRQ), Reference Signal Received Power (RSRP), Received Signal Strength Indicator (RSSI), and Signal-to-Interference-plus-Noise Ratio (SINR). These parameters provide critical insights into the quality and reliability of the wireless communication link.
- **Network-layer parameters:** Features under this category include the operational band, the type of radio access network (RAN), the identifier of the serving cell, and the mobile network operator. To ensure privacy and compliance with data protection standards, operator-related values are anonymized and coded as integers.
- **Mobility-related values:** This set captures the dynamic aspects of vehicular movement, including the vehicle's geographic position (latitude, longitude, and elevation), velocity, and acceleration. These parameters are vital for studying mobility's influence on network performance and QoS.
- **QoS parameters:** This category measures the end-user experience by including downlink (DL) and uplink (UL) throughput values, as well as network delay. These features are crucial for predictive QoS tasks and evaluating the performance of AI/ML models in real-world scenarios.

The diverse range of features in the dataset ensures a holistic representation of the interactions between the physical and network layers, vehicular mobility, and resulting QoS. By integrating these parameters, the dataset provides a robust foundation for investigating complex dependencies and facilitating advancements in predictive QoS for vehicular applications.

The vehicle's positioning data was collected using two advanced external sensors. First, the Global Navigation Satellite System (GNSS) data was acquired through a Ublox ZED-F9P Real-Time Kinematic (RTK) GNSS sensor. This sensor provided crucial geospatial

Table 4.2: Description of dataset values

Data source	Rate	Parameter	Unit
Teltonika RUTX50	1 Hz	timestamp	seconds (s)
		RSRQ	decibel (dB)
		RSRP	decibel (dB)
		RSSI	decibel (dB)
		SINR	decibel (dB)
		band	string
		RAN	string
		serving cell ID	integer
		delay (network ping)	milliseconds (ms)
		service status	boolean
		operator	integer
		DL throughput (ifstat in)	kilobytes per second (kb/s)
		UL throughput (ifstat out)	kilobytes per second (kb/s)
Ublox ZED-F9P	10 Hz	latitude	degrees
		longitude	degrees
		elevation	meters (m)
		GNSS mode	integer
Xsens MTi-680g	100 Hz	heading	degrees
		lateral acceleration	meters per second squared (m/s^2)
		longitudinal acceleration	meters per second squared (m/s^2)
		absolute acceleration	meters per second squared (m/s^2)
Vehicle CAN bus	72 Hz	absolute velocity	meters per second (m/s)
		longitudinal velocity	meters per second (m/s)

parameters, including latitude, longitude, altitude, and GNSS service quality. The positioning accuracy of the GNSS varied across the measurement sequences due to the dependence of RTK correction signals on mobile connectivity [129]. The GNSS service quality was categorized into three levels: Differential GNSS (DGNSS), RTK float, and RTK fix, reflecting the varying levels of positional precision.

To address inconsistencies in GNSS measurement accuracy, the vehicle's velocity data was captured through readings of wheel speeds obtained from the CAN bus, which were then converted into vehicle speed. For vehicle orientation, data was sourced from the Xsens MTi-680g inertia measurement unit, which provided robust inertial tracking and orientation information.

The mobile connectivity data was gathered using a Teltonika RUTX50 5G modem positioned on the vehicle's dashboard. Connectivity parameters were extracted using AT commands (where 'AT' denotes 'Attention'), which serve as a standardized set of Application Programming Interfaces (APIs) for interacting with cellular modems [130]. To ensure the collection of meaningful network downlink (DL) and uplink (UL) speed data, artificial bandwidth strain was introduced during the measurements by actively downloading large files over the mobile connection.

The UL and DL throughput metrics were measured at the application level using the `ifstat` API [131], providing accurate real-time data transfer rates. Similarly, network delay measurements were performed at the application level using the Linux `ping` utility [132]. The data saver software implemented this functionality by generating ping requests and recording the reception of the corresponding responses.

The software architecture for the data collection process is depicted in Fig. 4.2. Within the vehicle, internal network communication is facilitated via Ethernet, with the Teltonika router functioning as the sole gateway to the internet using a commercially available mobile subscription. Independent software drivers were developed to interface with the positioning sensors, the vehicle's CAN bus, and the Teltonika router. These sensor drivers collected measurements from their respective hardware components and published the

data over the vehicle's local network using the Data Distribution Service (DDS) protocol [133].

The measurements were aggregated by a central data saver application, which subscribed to all data streams on the network. Given the heterogeneity of sensors and devices, each producing data at varying rates, the data saver application recorded synchronized snapshots of the most recent values from each device at a consistent refresh rate of 1 Hz. This architectural design ensured seamless integration of diverse data sources and synchronized data logging.

The NordicDat dataset was gathered during several measurement sessions, consisting primarily of extended continuous driving runs. These drives featured speeds ranging from low urban velocities to highway speeds reaching 100 km/h. The data collection occurred in the Arctic rural regions of northern Finland, Norway, and Sweden, as shown in Fig. 4.3. While the dataset predominantly includes data from dynamic driving over long distances, it also captures several cross-border events where handovers occur between service providers in the respective countries.

Cross-border events provided an opportunity to study connection loss scenarios and their effects on network performance, including weakened GNSS signal modes due to interruptions in mobile connectivity. The routes were carefully selected to represent diverse Quality of Service (QoS) levels, encompassing areas with varying degrees of cellular coverage, including regions with poor connectivity and even complete service loss. This deliberate route selection ensured that the dataset contained rich, naturalistic data from realistic and challenging driving environments. In total, the dataset includes over 25 hours of measurements and covers nearly 1200 kilometers of driving. This comprehensive data collection effort was designed to support rigorous studies in predictive QoS for vehicular communication systems, especially in the context of remote and cross-border scenarios.

4.2.4 Data analysis and statistics

Predicting QoS values in vehicular networks poses significant challenges due to the inherently volatile nature of the network parameters involved [71]. One of the most critical influencing factors is the vehicle's location, as it encapsulates spatial effects related to physical layer attributes and the surrounding environmental characteristics [100]. Our dataset provides evidence to support these findings, highlighting the profound impact of spatial variations on QoS values.

The spatial effects heatmap (Fig. 4.3) illustrates how DL throughput fluctuates within the range of $[0, 20]$ Mbps across the entirety of the measurement campaign route. To further contextualize, the 25^{th}, 50^{th}, and 75^{th} percentile values for DL throughput during the campaign are recorded as 0.38 Mbps, 4.39 Mbps, and 12.03 Mbps, respectively. These variations underscore the critical role spatial factors play in the performance of vehicular communication networks.

An overview of the linear relationships between the pQoS metrics—DL and UL throughput, as well as delay—and corresponding network, spatial, and mobility features is shown in Fig. 4.4. While no substantial linear correlations are evident, certain patterns emerge. DL throughput exhibits a stronger dependency on mobility features (*e.g.,* vehicle velocity) and network-level parameters (*e.g.,* frequency band, RAN technology, serving cell, and operator). Conversely, UL throughput and delay are more influenced by physical layer parameters, such as RSRQ and SINR.

Temporal effects, which measure the influence of past QoS values on future predictions, reveal diverse behaviors across QoS metrics. As shown in Fig. 4.5, delay obser-

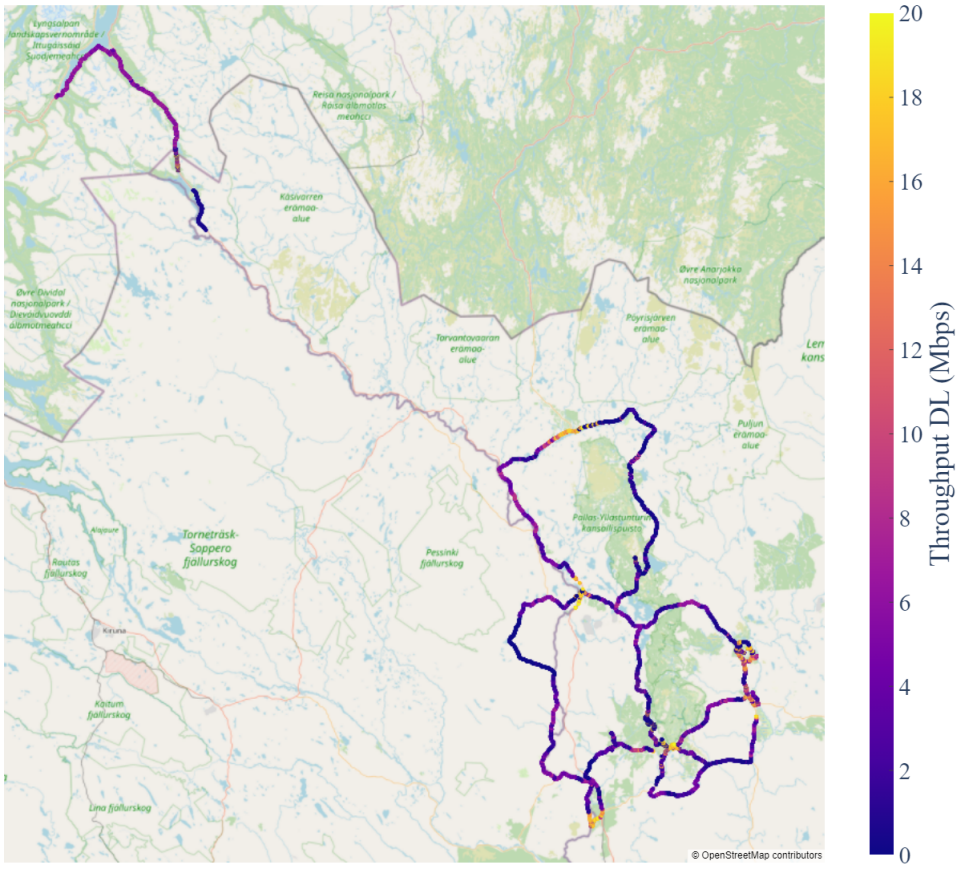


Figure 4.3: Spatial effects on QoS (DL Throughput) across the measurement area

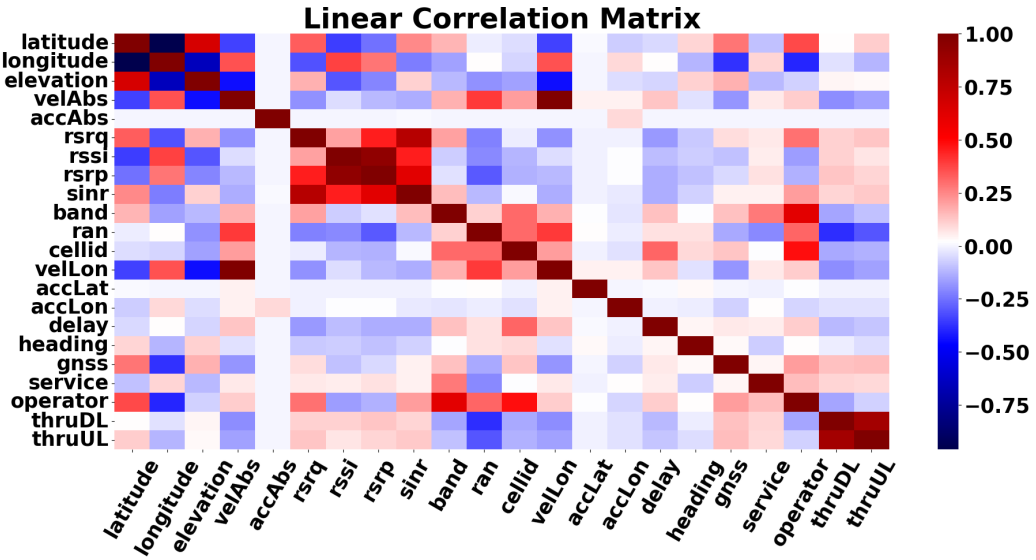


Figure 4.4: NordicDat feature correlation

vations exhibit minimal temporal autocorrelation, with values dropping below 0.5 for intervals exceeding 5 seconds, indicating negligible linear temporal dependencies. In contrast, throughput metrics display stronger temporal effects, with autocorrelation values exceeding 0.8 for intervals up to 50 seconds. These observations highlight that while temporal patterns are essential for predicting throughput, they are less relevant for delay predictions.

Moreover, our dataset validates the findings of prior research (detailed in Sec. 4.2.2), emphasizing the significant impact of three key factors on QoS metrics: a) RAN technology, b) roaming effects, and c) mobility patterns. Fig. 4.6 provides a comparative analysis that demonstrates the degradation in average throughput and an increased density of extreme values (outliers) under specific scenarios. Notably, these observations hold for the following bilateral comparisons: a) LTE versus 5G, b) roaming networks versus national networks, and c) highway versus urban mobility (velocities up to 25 km/h). Statistical analysis using the Mann–Whitney test confirms these distinctions, with P values approaching zero in all cases, underscoring the statistically significant impact of these factors on QoS outcomes.

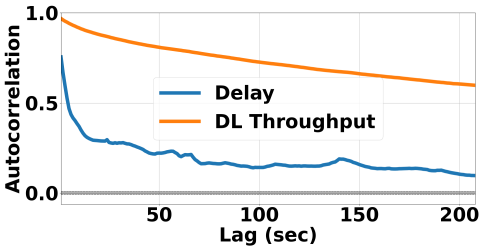


Figure 4.5: Temporal autocorrelation

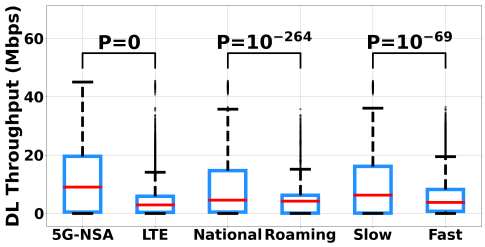


Figure 4.6: Feature dependencies

4.2.5 Potential usage and limitations

NordicDat exposes a wide array of features that include a) mobility-related metrics *e.g.,* position, speed, acceleration, b) physical layer cellular parameters *e.g.,* SNR, RSRQ, RSSI, c) network-level parameters *e.g.,* cell number, RAN, operator and finally d) QoS values, such as (appplication-level) throughput and delay. The resulted dataset is formulated as a time-series table and it can therefore be utilized to perform a series of relevant prediction tasks, besides pQoS. Relevant examples include handover (change of cell) prediction, vehicle trajectory prediction and driver intention classification.

The dataset's limitations are summarized as follows: 1) All features are obtained from the vehicle's devices. Though such an approach bypasses any MNO-related data confidentiality issues, it lacks information in regards to the overview of the network *e.g.,* cell capacity, total number of active UEs, distance to basestation, slicing policies, *etc.* [70]. 2) The 5G modem's available interfaces (see Sec. 4.2.3) do not provide support for additional features that could potentially enhance the accuracy of pQoS *e.g.,* resource blocks, Reference Signal Signal to Noise Ratio (RSSNR), Channel Quality Information (CQI), carriers number, coding schemes, *etc.* [71], [112]. 3) DL and UL throughput are measured via the Linux ifstat (application-level) API (see Sec. 4.2.3). As such, no information is given in regards to the transport, network, or link layer. 4) The cross-border locations under study mostly include highway road segments of low-traffic, as compared to an urban environment. The dataset therefore lacks instances of QoS degradation *e.g.,* strong interference incidents that are found in crowded areas, due to multiple parallel user transmissions. 5) All measurements are obtained from a single vehicle, therefore

scenarios that include multiple clients *e.g.,* distributed AI/ML tasks can only be applied via emulation (see Sec. 4.5).

4.3 Distributed pQoS

4.3.1 Problem statement and evaluation methodology

Motivated by the recent advances in distributed AI/ML, we have utilized our dataset to showcase the application of FL on pQoS and compare against its typical centralized AI/ML alternative. To the best of our knowledge that is the first attempt to apply distributed pQoS on a real-world public dataset.

To emulate an FL setup with multiple clients, we split our original dataset into 10 parts of equal size, each representing a single vehicle-client. We then run a series of pQoS training tasks using both centralized (CL) and distributed pQoS (FL) to predict a) delay and b) DL throughput, for a total of 400 experiments (2 AI/ML approaches $\times$ 100 repetitions $\times$ 2 QoS values).

For each client, data is split at a typical 80%-20% ratio [81]. In each training round, we select 8 clients for training and 2 for testing [101]. Round duration is set to 800 secs, for a total of 10 rounds per experiment. For the pQoS task we train a custom Long Short-Term Memory (LSTM) AI/ML model with the following characteristics: 22 input features (equal to the total features of NordicDat) and 8 output features *i.e.,* a prediction horizon of 8 sec. The LSTM model is tuned to the following parameters via grid-search: sliding window=75, hidden size=50, Min-Max normalization, decay=10^{-5}, Rectified Linear Unit activation and Mean Square Error loss function, batch size=64, learning rate=10^{-5}, epochs=50. QoS Prediction accuracy is evaluated using the *Root Mean Square Error (RMSE)* metric[81].

4.3.2 QoS Prediction

We firstly present two representative instances (for a single vehicle-client during a single experiment) of the inference results achieved by our FL model, in terms of DL throughput (see Fig. 4.7) and delay prediction (see Fig. 4.8). These qualitative representations suggest that the FL model is able to track the complex patterns and variations of the QoS values under study (throughput and delay), across time. The quantitative results that present the mean inference values of all clients, across all experiments, for all prediction horizons (from 1 up to 8 seconds ahead) are depicted in Fig. 4.9 and 4.10. As expected, longer prediction horizons are prone to larger prediction errors (RMSE) of the QoS value, as compared to shorter ones. Interestingly, the horizon's impact on throughput is much higher as compared to delay. Specifically for throughput, increasing the horizon from 1 to 3, 5 and 8 sec, results in an increase of RMSE by 5.55%, 10.09% and 15.07% (averaged across all rounds), respectively (see Fig. 4.9). For delay prediction on the other hand, the respective values are 2.36%, 3.67% and 4.11% (see Fig. 4.10).

Having said that, we fix the horizon value to 8 sec and compare the performance of distributed pQoS (FL) to that of the classical ML approach (CL), for both throughput and delay prediction tasks. We present the mean inference values and (shady) standard deviations of all clients, across all experiments in a per-round basis (see Fig. 4.11, 4.12). For throughput prediction, FL converges similarly to CL in the very few rounds of the experiments. For throughput prediction, CL outperforms FL by an average of 9.37% across all rounds (see Fig. 4.11). In fact, CL exhibits a maximum performance enhancement of

27.73% against FL. Interestingly though, FL achieves outperforms CL by 9.16%, during the experiment's last round. Unlike throughput, FL for delay prediction achieves a very similar performance to that of CL (see Fig. 4.12); on average CL outperforms FL by 2.08% across all rounds. FL tracks the performance of CL in the course of the time (rounds), even achieving better inference results (lower RMSE values) in certain instances. Overall our preliminary results suggest that distributed pQoS via FL can achieve similar accuracy levels to that of its centralized alternative, whilst preserving data privacy.

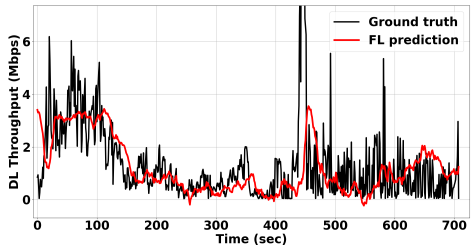


Figure 4.7: DL Throughput inference

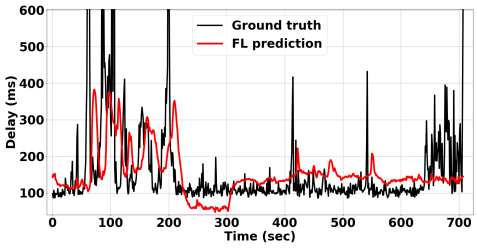


Figure 4.8: Delay inference

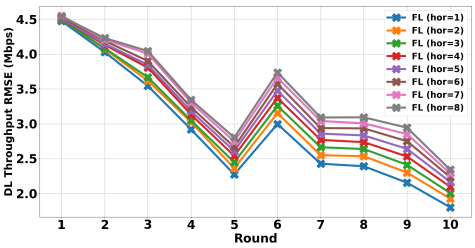


Figure 4.9: FL horizons: DL Throughput

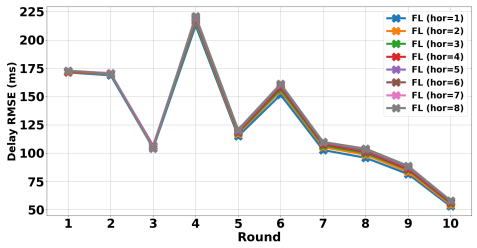


Figure 4.10: FL horizons: Delay

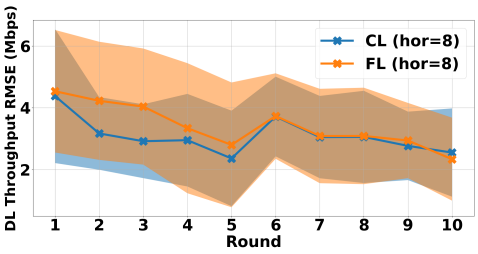


Figure 4.11: CL *vs.* FL (DL Throughput)

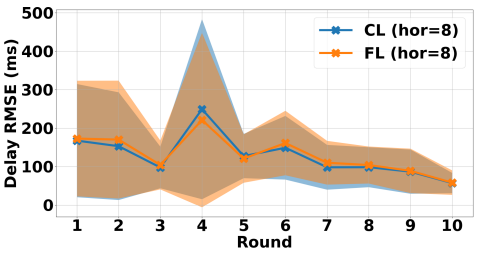


Figure 4.12: CL *vs.* FL (Delay)

4.4 5G Testbed implementation

4.4.1 Motivation

The next generation of mobile network technology is expected to significantly transform connectivity and data processing, offering global coverage, ultra-low latency, and exceptionally high data rates. The vision for beyond-5G networks revolves around the concept of *ubiquitous wireless intelligence* [134], which envisions a fully interconnected Internet-of-Everything landscape. This paradigm is driven by AI/ML, which is anticipated to enhance

human activities by enabling seamless reasoning, decision-making, and actuation across various systems and applications [135].

In this evolving landscape, UE plays a pivotal role, acting as the bridge between digital services and the end-user [136]. From smartphones and wearables in personal healthcare and infotainment to on-board units and collaborative robots in automotive and manufacturing industries, these devices are becoming increasingly sophisticated in terms of both communication and computational capabilities. As such, UEs are expected to extend the cloud-to-edge compute continuum, evolving into what is known as *Extreme-Edge computing*. This concept involves leveraging the spare resources of UEs to execute computational tasks, enabling more efficient and decentralized processing [137].

However, two primary challenges limit the full realization of this technological shift. First, these devices are typically under the control of the service provider, meaning they remain isolated from the broader cloud-to-edge continuum in 5G networks. Moreover, despite their growing computational and communication power, they are often treated as passive clients rather than active computing nodes [138]. Addressing these limitations presents several significant challenges [139], including: 1) managing the large number of Extreme-Edge devices (EEDs) and the coexistence of various vertical services, 2) ensuring reliable connectivity for EEDs, which may experience issues such as connection unavailability or dropouts, 3) handling resource constraints like processing capacity, storage, memory, and battery life, 4) navigating the multi-stakeholder environment, as EEDs are rarely under the control of mobile network operators (MNOs), and 5) overcoming interoperability challenges due to hardware diversity.

While these challenges remain largely under-explored in existing literature, current approaches either focus on theoretical models, often relying on simulations that lack real-world applicability [138], [140], or are deployed in constrained local environments [135], [136], overlooking the inherent volatility of network connectivity. To address these gaps, we adopt a systems-oriented approach by designing and implementing the Extreme-Edge Orchestrator (EEO), a management and orchestration (M&O) framework that integrates Extreme-Edge resources within the 5G ecosystem. Our solution leverages cloud-native tools to monitor network resources and perform lifecycle management (LCM) of (containerized) network services running on EEDs. LCM decisions, such as task initiation, selection of EEDs for task execution, and graceful task termination, are driven by user-defined policies, establishing closed-loop control mechanisms. These loops incorporate several criteria, including: a) device and utilization characteristics (*e.g.,* processing capacity), b) network resource consumption parameters (*e.g.,* energy costs), and c) application-level features (*e.g.,* data availability), which may involve AI/ML models.

In contrast to the majority of simulation-based research in this area, our proposed solution is deployed on an operational 5G testbed and evaluated through extensive real-world experiments involving both mobile and static EEDs in diverse (indoor and outdoor) scenarios. Our contribution is two-fold: 1) we introduce, design, and develop a novel solution for Extreme-Edge orchestration, along with the corresponding system architecture that extends the cloud-to-edge continuum to the Extreme-Edge, and 2) we evaluate our solution on an experimental 5G testbed through a relevant use case for the automotive sector: predicting network QoS values using distributed AI/ML. Our analysis demonstrates the effectiveness of the multi-criteria EED selection mechanism in scenarios where including EEDs yields marginal benefits at the application layer, particularly for AI/ML tasks.

4.4.2 Related work

Service orchestration at the Extreme-Edge has recently gained significant attention as a means to extend cloud-native technologies throughout the compute continuum. This transition has stimulated the development of novel architectures [140] and intelligent algorithms [138], although these efforts are still largely in the simulation phase, with limited real-world deployment and validation.

At the system level, several attempts have been made to experimentally assess such solutions in prototype testbeds, aiming to leverage resource-constrained devices, such as IoT devices, as compute nodes. For instance, an experimental comparison of existing edge orchestration tools is presented in [139], which evaluates these solutions in terms of scalability and service instantiation time for EEDs. With the increasing processing demands driven by AI/ML applications, many studies have shifted focus towards orchestrating AI/ML pipelines at the Extreme-Edge, either for inference [136] or training [135] phases. Additionally, some propose managing AI/ML pipelines using the microservice paradigm [141]. While these studies provide valuable insights into the computational capabilities of EEDs, they are primarily evaluated in local environments or on connected Virtual Machines (VMs), which often overlook key factors such as network connectivity, device interoperability, and hardware dependencies that are critical in real-world deployments.

An emerging area of research focuses on the development of new or the adaptation of existing cloud-native tools specifically designed for orchestration at the Extreme-Edge. For example, in [137], a lightweight container system for Extreme-Edge computing is introduced, with evaluations conducted across devices featuring diverse architectures and operating systems (OS), connected via Hypertext Transfer Protocol (HTTP). Similarly, in [142], a novel architecture tailored for Machine Learning (ML) deployment in the Extreme-Edge is proposed and experimentally validated in a 4G-enabled testbed using Unmanned Aerial Vehicles (UAVs) as UEs, offering insights into the performance and resource consumption trade-offs. However, these solutions are limited in that they operate primarily at the container orchestration level, failing to consider the application-specific requirements and dependencies that often arise in dynamic environments.

In contrast, our proposed approach introduces a service LCM layer that enables adaptive service deployment at runtime. This layer is governed by user-defined logic that incorporates both resource-related and application-specific criteria, allowing for a more flexible and efficient orchestration framework at the Extreme-Edge. By addressing both the computational and network-related challenges in real-world scenarios, our solution provides a comprehensive approach to orchestrating services in highly dynamic and resource-constrained environments.

4.4.3 Experimental testbed setup

In this section we provide the details of the software and hardware components that comprise the 5G testbed for our experimental analysis.

System architecture

Our implemented software stack is based on Microk8s [143], a lightweight Kubernetes variant, as shown in Fig. 4.13. Microk8s enables the configuration and M&O of a cluster of computing nodes (workers) that run containerized applications. In our context, these workers can refer to the compute resources on the EEDs or on the Multi-Access Edge Computing (MEC) domain. The Microk8s controller resides on the MEC, providing the

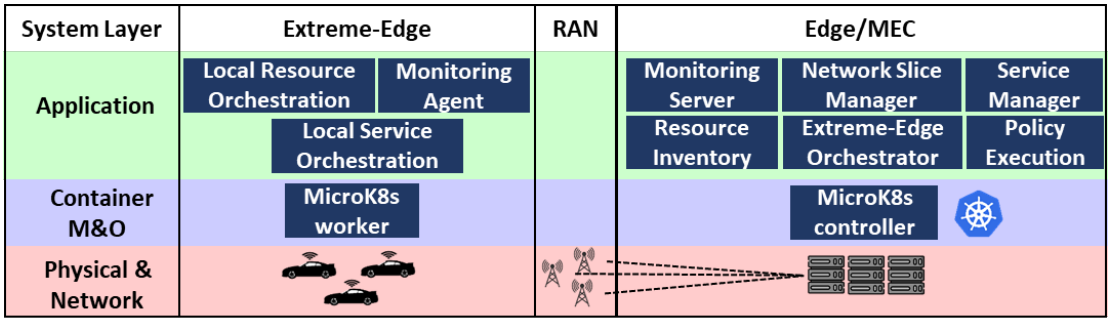


Figure 4.13: System architecture and software stack

interfaces to manage the processing resources within the cluster. All available services are cataloged within the MEC's Service Manager, a module that contains software artifacts (or a reference of them) and descriptors to deploy a (set of) service(s) that provide functionalities to verticals, over the 5G infrastructure. The Network Slice Manager handles network resource allocation during deployment, addressing the service requirements specified in the descriptor and establishing the corresponding connection with the EEDs over the Radio Access Network (RAN). On the EED side, the Local Resource Orchestration and Local Service Orchestration modules manage network and service deployment, respectively.

Once deployed, orchestration actions for the service runtime management are conducted through Policy Execution. Policies are expressed as a set of rules and actions, defined as Service Level Agreement (SLA) expressions that represent the desired service-level state. Rules are associated with the deployed service graphs at any given moment. Policy execution involves a series of Representational State Transfer (REST) interfaces that facilitate the manipulation of service policy rules. These interfaces support fundamental operations *i.e.,* Create, Read, Update, and Delete (CRUD), enabling actions like creating new policy rules, updating or deleting existing ones and retrieving policy rule information. Additionally, the interfaces provide search functionalities based on criteria such as policy name or identification number. The MEC also hosts a monitoring entity that collects monitoring data from the underlying virtualized infrastructure. This entity supports core services, including policies and is responsible for data collection via active monitoring probes. This functionality is based on Prometheus [143] and includes: a) a Monitoring Agent on each worker that collects telemetry metrics: total and utilized system resource capacity (random access memory - RAM, disk/storage, central processing unit - CPU, and graphics processing unit - GPU), number of CPU cores, UL/DL network usage, fan and temperature readings, power consumption, location (latitude/longitude) and application data volume; and b) a Monitoring Server located in the MEC that records and stores monitoring data as time-series in the Resource Inventory module.

LCM of processing tasks in the (Extreme-)Edge is performed by the EEO, located in the MEC. These tasks span from typical automotive services *e.g.,* video streaming to computationally-intensive processes *e.g.,* on-device/distributed AI/ML. The EEO enables: 1) service configuration during initialization, 2) worker selection at runtime to run the service *i.e.,* add a new or remove an existing worker and 3) graceful service termination. To enable these functionalities, the EEO exposes a REST interface that allows the end-user *e.g.,* the service provider, to provide a series of input criteria (see Table 4.3): a) generic characteristics (device characteristics, run-time resource utilization parameters, location filters, termination criteria) and b) application-specific criteria with support for AI/ML tasks (configuration, data-related criteria, AI/ML performance metrics). Once the criteria

are set, the EEO functions as follows: it continuously monitors both the service and the available workers, via the Monitoring Server. It thereafter performs a matchmaking to determine if the specified user-defined criteria are satisfied. Based on the results of the matchmaking process, the EEO infers appropriate actions; for instance, it may decide to remove (scale in) an existing worker if its RAM utilization exceeds a predetermined threshold, or to add a new worker (scale out) that has sufficient storage capacity. These inferred actions are then converted into policy commands (rules) and propagated to the Policy Execution module for implementation.

The aforementioned baseline functionalities enable a series of LCM policies to be enforced, including: 1) **Resource-aware task execution** - tuning of critical task parameters *e.g.,* duration, number of EEDs, *etc.,* to balance task performance with resource consumption, 2) **Task geo-fencing** - restricts task execution to EEDs within a defined area, enhancing privacy, 3) **Resource provision** - excludes heavily utilized EEDs to reduce risk of failure and improve robustness, 4) **Load balancing** - ensures fair distribution of computational load across the EEDs, 5) **Optimal task stopping** - timely terminate a task based on the optimal stopping theory [144] for efficiency, 6) **Resource-aware AI/ML hyper-parameter tuning** - configure AI/ML parameters with consideration of resource costs, 7) **Smart AI/ML scheme selection** - schedules AI/ML training in a centralized, distributed or hybrid mode, based on the expected performance and resource costs, and 8) **Data-efficient AI/ML client selection** - selects EEDs with sufficient data volume to enhance training efficiency. Several of these features are demonstrated in the forthcoming Sec. 4.5.

Table 4.3: Extreme-Edge Orchestrator (EEO) input criteria

Type	Criterion	Description	Type	Criterion	Description
Device selection: characteristics	Access rights	Service provider ownership of devices (private) or independent status (public)	Device selection: AI/ML condition	Dataset size	Min application data volume (MB)
	GPU availability	True/False		Local train loss	Min client AI/ML convergence (%)
	GPU capacity	Min number of GPU cores		Local test loss	Min client AI/ML accuracy (%)
	CPU capacity	Min number of CPU cores	AI/ML configuration	ML scheme	Centralized, Distributed or Hydrid AI/ML
	RAM capacity	Min total RAM (GB)		Clients per round	Min client devices for AI/ML
	Storage capacity	Min total disk capacity (GB)		Number of epochs	Hyper-parameter setting
Device selection: Runtime resource utilization	CPU utilization	Max CPU usage-to-capacity ratio (%)		Batch size	Hyper-parameter setting
	GPU utilization	Max GPU usage-to-capacity ratio (%)		Learning rate	Hyper-parameter setting
	RAM utilization	Max RAM usage-to-capacity ratio (%)		ML model	AI/ML model type *e.g., LSTM*
	Temperature	Max device temperature (°C)		Aggregation algorithm	(For distributed AI/ML) Mean/Median type
	Occupied storage	Max current-to-total disk capacity ratio (%)		Training rounds	Number of training cycles
	Energy availability	Min remaining battery level (%)	Task termination	Global train loss	Max AI/ML convergence of all clients (%)
	Uplink bandwidth	Max uploaded data volume (MB)		Global test loss	Max AI/ML accuracy of all clients (%)
	Downlink bandwidth	Max downloaded data volume (MB)		Total energy	Max energy consumed across all devices (KJ)
Device selection: Location filters	Area of interest (AoI)	Bounding box (latitude and longitude for each corner)		Total data volume	Max data consumed across all devices (MB)
	Devices in area	Max number of devices		AI/ML efficiency	Optimal stopping for global accuracy w.r.t. total energy consumed

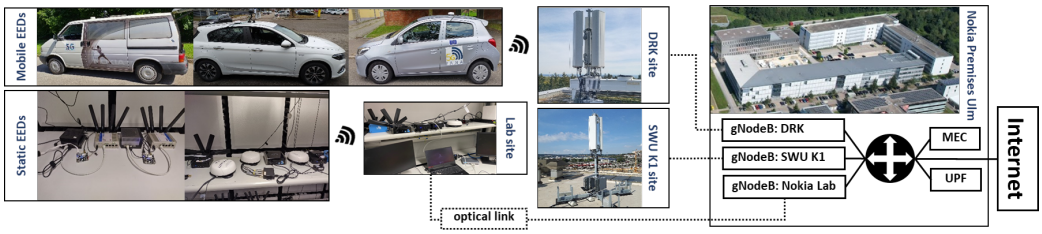


Figure 4.14: Testbed hardware and network setup

Hardware and network setup

Network connectivity across the testbed is provided by a standalone 5G test network in Ulm, Germany, operated by Nokia. As depicted in Fig. 4.14, the testbed comprises: a) the core and edge network, b) the RAN and c) several EEDs. The hardware specifications for all involved devices are presented in Table 4.4. The core and edge network infrastructure, hosted at Nokia's Ulm facilities, includes a) gNodeB central units, b) a MEC server and c) the user plane functions (UPFs), all of which are connected via a switch. The switch provides access to external Data Networks (DNs) *e.g.,* the Internet and to 5G control plane functions *e.g.,* Access and Mobility Management Function (AMF), Session Management Function (SMF), *etc.,* located at Nokia's site in Finland. User data forwarding between the gNodeBs and the DNs is handled by the UPFs, while the MEC server provides compute resources, to support services at the network edge.

The RAN consists of three antenna sites, namely DRK Ulm, SWU K1 Ulm and Nokia lab picocell, each connected via optical links to its respective gNodeB. Each site supports three sector cells on the n38 band, covering a frequency range of 2575–2615 MHz. Time Division Duplex is employed to support both uplink (UL) and downlink (DL) communication within the 40 MHz frequency range, with an UL/DL split set at 30/70. The DRK and SWU K1 Ulm sites are set for outdoor deployment and support mobile EEDs, while the Nokia lab site is designated for indoor deployment, primarily supporting static EEDs. Each EED is equipped with a 5G modem to allow for wireless connectivity and possesses diverse processing capabilities—including variations in storage, memory, and processing power (see Table 4.4). Mobile EEDs are deployed within vehicles powered by an external power source, whereas static EEDs are located within the Nokia lab.

Table 4.4: Testbed device specifications

	MEC Server	EED 1	EED 2	EED 3	EED 4
Device type	Xeon E5-2680	Besstar UM560 DeskMini	Besstar UM560 DeskMini	NVIDIA Jetson AGX Orin	NVIDIA Jetson AGX Xavier
CPU type	AMD Ryzen 9 5900X	AMD Ryzen 5 5625U	AMD Ryzen 5 5625U	Arm Cortex-A78AE v8.2	512-core NVIDIA Volta
CPU cores	12	12	12	12	8
GPU type	NVIDIA RTX 3080 TI	AMD ATI 04:00.0 Barcelo	AMD ATI 04:00.0 Barcelo	NVIDIA Ampere (2048 cores)	NVIDIA Carmel Arm®v8.2
RAM (GB)	128	16	14	32	32
Storage	SSD (2 TB), HDD (8 TB)	SSD (500 GB)	SSD (500 GB)	SSD (60 GB)	SSD (30 GB)
Operating system	Ubuntu 22.04 LTS	Ubuntu 22.04.4 LTS	Ubuntu 22.04.4 LTS	Ubuntu 20.04.6 LTS	Ubuntu 20.04.6 LTS
5G Modem	None	Quectel RM500Q-AE	Quectel RM500Q-AE-VA	Teltonica RUTX50	Teltonica RUTX50

4.4.4 Implementation of distributed pQoS

To showcase the usage of the EEO, we have selected a relevant use-case of the automotive vertical, namely distributed pQoS. Our implementation employs the FL paradigm to train a pQoS model in a distributed fashion. FL involves two main components; an aggregation server (deployed in the MEC in our setup) and training agents/clients (deployed on the EEDs). The pQoS model training occurs in consecutive cycles. In each cycle, the

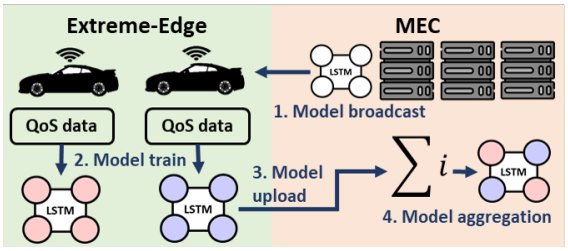


Figure 4.15: Distributed pQoS using Federated Learning

aggregation server selects a group of EEDs and broadcasts the global pQoS model, as depicted in Fig. 4.15. The selected EEDs then perform local model training, using their collected data. The local models are then uploaded to the MEC server, where the aggregation server combines (aggregates) them into an updated global model. This process is repeated for several times, until the model converges. The implementation of FL and its associated algorithms are based on Flower [145], a Python-based open-source framework for FL pipelines.

4.5 5G Testbed experiments

4.5.1 Evaluation methodology

For the evaluation of the EEO, we are using the experimental 5G testbed and a total of four EEDs. The EEDs serve as FL clients to perform distributed pQoS and specifically round-trip time (RTT) prediction. The EEDs constantly collect: a) location (latitude, longitude) data via a Global Positioning System (GPS) sensor and b) RTT data, via active measurements, along with their respective timestamps. The collected time-series data is used to train a Long Short-Term Memory (LSTM) pQoS model, given the LSTM's inherent ability to capture spatio-temporal dependencies [145]. The LSTM model is tuned to the following parameters via grid-search [145]: Layers=2 (1 LSTM with Tanh activation, 1 Dense with Linear activation), Sliding window=400, Horizon=1, Min-Max normalization, Tilted loss function, Batch size=32, Learning rate=10^{-2}, Epochs=5.

We initially validate the device selection capability of the EEO in presence of user mobility; two out of four EEDs are deployed inside vehicles, while the remaining are placed in the Nokia lab. On the EEO side, we set the following location filter criteria: a) Nokia premises, as the designated area of interest (AoI) and b) maximum of three devices in the area (see Table 4.3). Then, the vehicles are instructed to move sequentially in and out of the AoI. The demonstration results [146] suggest that the EEO successfully tracks these events and timely adjusts the FL pQoS service, by selecting (adding/removing) the respective EEDs, according to the criteria set. For further evaluation we run in-lab experiments, using the indoor 5G deployment (see Sec. 4.4.3) and a total of four static EEDs to perform FL pQoS. The assessment is based on the following metrics: 1) **LSTM Model (Training) Convergence** (in ms for RTT prediction) using the Mean Absolute Error (MAE) metric [145], 2) **QoS Prediction (Testing) Accuracy** (ms) via the Root Mean Square Error (RMSE), a relevant metric for pQoS [71], 3) the **Aggregated Energy** (in KJ) and 4) **Data Volume** (in MB) consumed by the EEDs during training, as recorded by the EEO's monitoring service. We repeat each experiment for 5 times and present the mean and standard deviation values for all measures quantities.

4.5.2 Orchestration at the Extreme-Edge

Effect of the number of training rounds

Initially, all four EEDs are employed to train the pQoS model via FL. In the EEO, we set a variable number of total training rounds (see Table 4.3) within the range of [10, 50] to assess their impact on the model's performance with respect to (w.r.t.) the underlying network resource consumption. As depicted in Fig. 4.16, increasing the number of training rounds leads to a reduction in prediction errors (MAE and RMSE, respectively), indicating improvements in both convergence and accuracy. However, the enhancement beyond 30 rounds is marginal (less than 10%). At the same time, allowing for more training rounds results in a linear increase in network resource consumption (see Fig. 4.17). In fact, when comparing to 30 training rounds, the use of 40 and 50 training rounds results in increases by 34% and 69% in data volume as well as 33% and 65% in energy consumption, respectively. The EEO allows for tuning the number of rounds and control the trade-off between model performance and resource consumption. In our study, we select 30 rounds for our task, maintaining this value throughout the remainder of our experiments, to achieve adequate accuracy (less than 0.3 ms RMSE for automotive applications [104]), while also considering the efficiency of resource consumption.

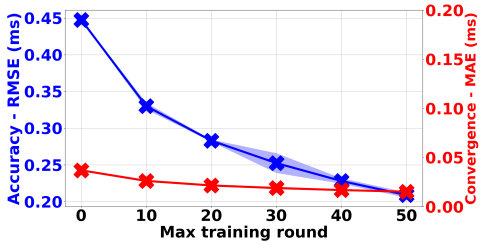


Figure 4.16: Number of rounds: Accuracy

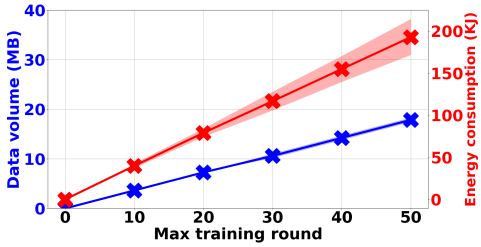


Figure 4.17: Number of rounds: Resource cost

Effect of the number of participating EEDs

Having set the number of training rounds, we now examine the impact of varying the number of (FL) clients, specifically using 2, 3, and 4 EEDs (see Table 4.3). As shown in Fig. 4.18, the accuracy differences during the first 10 training rounds (approximately 30 minutes) are minimal, with less than a 2% variation in accuracy among the three configurations (calculated as the mean value for the first 30 minutes). By the end of the experiments, utilizing 3 and 4 clients leads to an accuracy increase of up to 4% compared to training with 2 clients. This also increases the involved bandwidth costs by 58% and 117% (see Fig. 4.19) and energy costs by 49% and 103% (see Fig. 4.20), respectively. Our results suggest that while the number of participating EEDs affects model performance, it primarily impacts the associated costs. As such, the vertical service provider can define policies and adjust the number of selected EEDs accordingly, via the EEO's monitoring and control *i.e.,* selection functionality. For the upcoming scenarios, we set this parameter to 3, as this setting achieves similar accuracy to that of 4 clients, while incurring substantially lower costs.

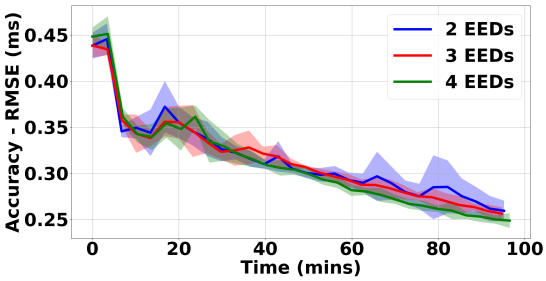


Figure 4.18: Number of EEDs: Accuracy

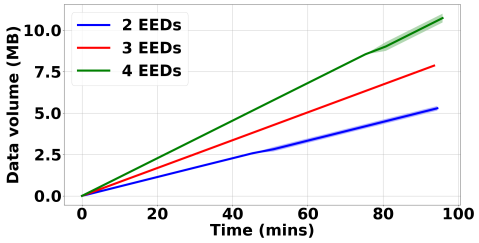


Figure 4.19: Number of EEDs: Data volume

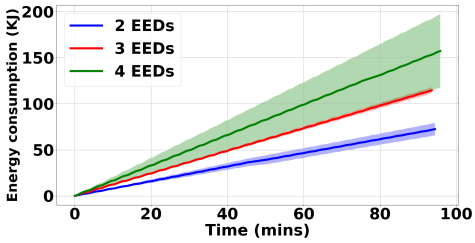


Figure 4.20: Number of EEDs: Energy

CPU and RAM stress scenarios

We proceed by testing multiple service scenarios under the assumptions that the EEDs: 1) run an FL pQoS service and 2) are periodically tasked with a higher-priority automotive service *e.g.,* High Definition map update. In such a scenario stress will be eventually imposed on the EEDs' CPU and RAM capacity. To simulate these conditions, we use stress-ng [147]; fluctuations of CPU utilization (and therefore availability) of the EEDs over time are shown in Fig. 4.21. We then configure the EEO as follows: the high priority automotive service runs on all EEDs upon request, while FL (being of lower priority) runs on the EEDs that have less than 30% CPU utilization (see Table 4.3). We compare against the Vanilla case (without the EEO), where all EEDs are greedily selected for each service. The results in the training performance are presented in Fig. 4.22. By smartly selecting the EEDs with high CPU availability, the FL model in the EEO case is trained faster, with a reduced average round duration by 24%, compared to the Vanilla case. On the contrary, the EEDs in the Vanilla case are selected as FL clients regardless their CPU load, thereby extending the duration of each training round and the completion time of the FL task, as a whole. On top, the total energy expenditure in the Vanilla case is 24% higher compared to the EEO case (see Fig. 4.23), consistent with the linear relationship between resource utilization and energy consumption, presented in Fig. 4.20. When varying the RAM instead of the CPU, the results indicate even greater differences between configurations. As shown in Fig. 4.24, the EEO's smart client selection facilitates smooth training, resulting in the development of a pQoS model with an accuracy of 0.25 ms (RMSE) at a total energy cost of approximately 120 KJ, consistent with the outcomes shown in Fig. 4.18 and Fig. 4.20. In the Vanilla case however, excessive RAM overload leads to training failure and premature termination of the training process (see blue horizontal line in Fig. 4.24).

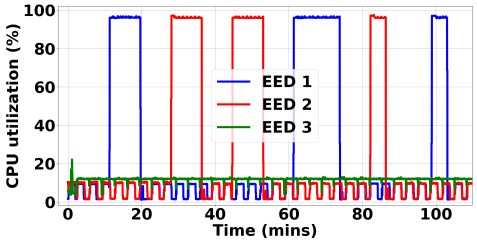


Figure 4.21: CPU stress: Setup

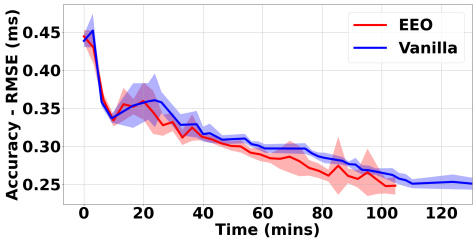


Figure 4.22: CPU stress: Accuracy

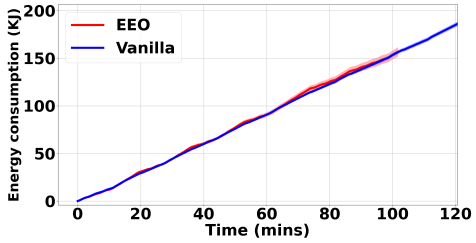


Figure 4.23: CPU stress: Energy

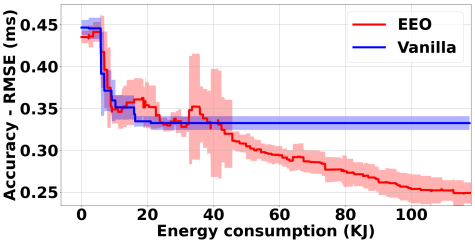


Figure 4.24: RAM stress: Performance test

4.6 Conclusions

In this chapter, we have presented two practical implementations of the theoretical concepts explored in earlier chapters, focusing on FL in distributed CCAM environments from a systems perspective.

Firstly, we introduced NordicDat, a comprehensive dataset collected over two weeks across the cross-border regions of Finland, Norway, and Sweden. This dataset includes a wide range of features related to cellular QoS values, such as throughput and delay, network characteristics like cell and operator information, and vehicle kinematics, including speed and location data. These features make the dataset highly valuable for training both classical and distributed AI/ML models across a variety of use cases, such as trajectory prediction, handover prediction, and, importantly, QoS prediction. The latter is recognized as a crucial enabler for the future development of automotive applications. Through our data analysis, we identified non-linear correlations between various features and the QoS values, highlighting that factors such as roaming, vehicle speed, and radio access technology significantly influence the QoS patterns over time and space. Additionally, we demonstrated the use of distributed AI/ML for QoS prediction, specifically for throughput and delay, which, to our knowledge, marks the first real-world demonstration of distributed predictive QoS (pQoS) on real-world data.

Next, to address the critical limitations in integrating terminal-side (Extreme-Edge) devices into the operational control framework of 5G networks, we introduced the Extreme-Edge Orchestrator (EEO), our novel cloud-native management and orchestration solution. The EEO combines network resource provisioning with multi-criteria, user-defined logic to enable efficient service orchestration for containerized applications running on Extreme-Edge devices. Our implementation of the system was extensively evaluated on an operational 5G testbed, focusing on distributed AI/ML tasks. The experimental results showed that the EEO improves resource efficiency by optimizing key parameters, such as task duration, and reduces task completion time by 25% under high computational load when compared to baseline methods. These findings suggest that managing Extreme-

Edge devices in next-generation networks can lead to more efficient and effective service delivery. Moreover, they point to several promising avenues for future research, including large-scale evaluations involving a greater number of end devices and extending the EEO's multi-criteria logic through the use of rule engines, which will be explored in subsequent work.

In summary, this chapter has contributed to advancing both theoretical and practical aspects of distributed AI/ML in the context of CCAM, with a particular focus on enhancing the integration of Extreme-Edge devices and improving QoS prediction capabilities in real-world automotive environments.

5

Conclusion

5.1 Key takeaways

This thesis presented our research on the convergence of two key advancements in next-generation networks: (a) the increasing utilization of client device processing capabilities within a unified cloud-to-edge network and compute continuum, and (b) the rise of Distributed AI/ML, particularly Federated Learning (FL), as a means of enabling intelligent decision-making for connected systems. Unlike previous theoretical studies, our focus has been on practical applications, specifically in the demanding domain of Connected and Cooperative Automated Mobility (CCAM), where both trends were relevant but are also coupled with complex requirements (e.g., safety).

Our work addressed several previously under-explored questions: (a) Which ML scheme, Centralized Learning (CL) or Distributed Learning/FL, is more efficient from a network perspective? (b) What roles did various factors such as models, energy consumption, and concept/model drift play in influencing these schemes? (c) What were the implementation challenges, and how could they be addressed when deploying these schemes in real-world environments?

To answer these questions, we presented the theoretical concepts, simulation environment, and the results of our study, which provided valuable insights on: a) The system parameters that influenced the efficiency of each ML scheme, in terms of both ML performance (accuracy) and underlying network and energy resource consumption. b) The resource consumption of DML/FL, considering AI/ML and network parameters. c) Trade-offs and considerations for selecting the appropriate ML scheme. d) The detrimental impact of concept drift in volatile network and vehicular environments for FL. e) Techniques for detecting and mitigating concept drift in distributed, resource-constrained environments, while minimizing network resource usage. f) The results of a measurement campaign and proof-of-concept implementation for predictive Quality of Service (pQoS). g) A management and orchestration framework for deploying multiple AI/ML and FL services in the automotive domain, demonstrated on a commercial-grade 5G testbed with multiple vehicles as mobile client devices.

Future research could expand on our concept drift management framework by incorporating dynamic adaptation through AI/ML techniques. Along with our findings on ML scheme selection, this could evolve into a comprehensive end-to-end ML Operations (MLOps) framework, capable of dynamically adapting to drift and adjusting functionality or switching between ML schemes based on user requirements.

Ultimately, our orchestration solution provided a valuable toolkit for managing the lifecycle of AI/ML and FL services and handling compute resources on Internet of Things

(IoT) devices. This could be integrated with the aforementioned algorithmic framework to enable continuous monitoring and management of multiple FL tasks across networks of mobile devices, vehicles, and IoT systems.

5.2 Publications

5.2.1 Journal/Magazine Articles

- **Drainakis, G.**, Pantazopoulos, P., Katsaros, K. V., Sourlas, V., Amditis, A., & Kaklamani, D. I. (2023). From centralized to Federated Learning: Exploring performance and end-to-end resource consumption. Elsevier Computer Networks, 225, 109657.

- Katsaros, K. V., Liotou, E., Moscatelli, F., Rokkas, T., **Drainakis, G.**, Bonetto, E., Brevi, D., Klonidis, D., Neokosmidis, I., & Amditis, A. (2022). Enabling far-edge intelligent services with network applications: the automotive case. IEEE Internet of Things Magazine, 5(4), 122-128.

5.2.2 Peer-reviewed Conference/Workshop papers

- **Drainakis, G.**, Pantazopoulos, P.,Katsaros, K. V., Sourlas, V., Xirofotos, T., Baganal-Krishna, N., Rizk, A., Horvath, R., Scivoletto, G., Amditis, A., & Kaklamani, D. I. Service Orchestration at the Extreme-Edge: An Experimental Investigation Over a 5G Testbed. In ICC 2025 IEEE International Conference on Communications (*under peer review*)

- Miekkala, T., Pyykonen, P., **Drainakis, G.**, Pantazopoulos, P., Muller, T., Katsaros, K. V., Sourlas, V., Amditis, A., & Kaklamani, D. I. NordicDat: A Cross-Border Predictive QoS Dataset. In GLOBECOM 2024 IEEE Global Communications Conference

- **Drainakis, G.**, Pantazopoulos, P., Katsaros, K. V., Sourlas, V., Amditis, A., & Kaklamani, D. I. (2024, June). Distributed Predictive QoS in Automotive Environments under Concept Drift. In 2023 IFIP Networking Conference (IFIP Networking). IEEE.

- Sourlas, V., Rizk, A., Katsaros, K. V., Pantazopoulos, P., **Drainakis, G.**, & Amditis, A. (2021, September). A Distributed ML Framework for Service Deployment in the 5G-based Automotive Vertical. In 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom) (pp. 246-251). IEEE.

- **Drainakis, G.**, Pantazopoulos, P., Katsaros, K. V., Sourlas, V., & Amditis, A. (2021, July). On the resource consumption of distributed ml. In 2021 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN) (pp. 1-6).

- **Drainakis, G.**, Pantazopoulos, P., Katsaros, K. V., Sourlas, V., & Amditis, A. (2021, May). On the distribution of ML workloads to the network edge and beyond. In IEEE INFOCOM 2021-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) (pp. 1-6).

- **Drainakis, G.**, Katsaros, K. V., Pantazopoulos, P., Sourlas, V., & Amditis, A. (2020, November). Federated vs. centralized machine learning under privacy-elastic users: A comparative analysis. In 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA) (pp. 1-8).

5.2.3 Under preparation/submission

- **Drainakis, G.**, Pantazopoulos, P., Katsaros, K. V., Sourlas, V., Amditis, A., & Kaklamani, D. I. Federated Learning in Automotive Environments under Concept Drift (Journal follow-up of the IFIP Networking paper, under submission)

Bibliography

[1] P. Wang *et al.*, "Introduction: Advances in iot research and applications," *Information Systems Frontiers*, vol. 17, pp. 239–241, 2015.

[2] Z. Qadir *et al.*, "Towards 6g internet of things: Recent advances, use cases, and open challenges," *ICT express*, vol. 9, no. 3, pp. 296–312, 2023.

[3] A. Shahraki *et al.*, "When machine learning meets network management and orchestration in edge-based networking paradigms," *Journal of Network and Computer Applications*, vol. 212, p. 103558, 2023.

[4] "Ieee standard for local and metropolitan area networks: Overview and architecture," *IEEE Std 802-2014 (Revision to IEEE Std 802-2001)*, pp. 1–74, 2014.

[5] A. Yegin *et al.*, "Lorawan protocol: specifications, security, and capabilities," in *LP-WAN Technologies for iot and m2m applications*. Elsevier, 2020, pp. 37–63.

[6] A. Al-Dulaimy *et al.*, "The computing continuum: From iot to the cloud," *Internet of Things*, vol. 27, p. 101272, 2024.

[7] L. A. Haibeh *et al.*, "A survey on mobile edge computing infrastructure: Design, resource management, and optimization approaches," *IEEE Access*, vol. 10, pp. 27 591–27 610, 2022.

[8] H. Kim *et al.*, "Mobile edge computing enabler layer: Edge-native application architecture for mobile networks," *IEEE Communications Standards Magazine*, 2023.

[9] H. Peng *et al.*, "Evaluating emerging ai/ml accelerators: Ipu, rdu, and nvidia/amd gpus," in *Companion of the 15th ACM/SPEC International Conference on Performance Engineering*, 2024, pp. 14–20.

[10] H. Kimm *et al.*, "Performance comparision of tpu, gpu, cpu on google colaboratory over distributed deep learning," in *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 2021, pp. 312–319.

[11] T. Tan and G. Cao, "Efficient execution of deep neural networks on mobile devices with npu," in *Proceedings of the 20th International Conference on Information Processing in Sensor Networks (Co-Located with CPS-IoT Week 2021)*, 2021, pp. 283–298.

[12] M. S. Allahham *et al.*, "On the modeling of reliability in extreme edge computing systems," in *2022 5th International Conference on Communications, Signal Processing, and their Applications (ICCSPA)*. IEEE, 2022, pp. 1–6.

[13] P. Basaras *et al.*, "Experimentally assessing deployment tradeoffs for ai-enabled video analytics services in the 5g compute continuum," in *2023 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*. IEEE, 2023, pp. 99–104.

[14] M. Murshed *et al.*, "Machine learning at the network edge: A survey," *arXiv preprint arXiv:1908.00080*, 2019.

[15] T. Wu *et al.*, "A brief overview of chatgpt: The history, status quo and potential future development," *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 5, pp. 1122–1136, 2023.

[16] M. Hussain, "Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection," *Machines*, vol. 11, no. 7, p. 677, 2023.

[17] A. Kolmogorov, "On the representation of continuous functions of several variables by superpositions of continuous functions of lesser variable count," in *Dokl. Akad. Nauk SSSR*, vol. 108, no. 2, 1956.

[18] W. Hong *et al.*, "Optimal design of hybrid federated and centralized learning in the mobile edge computing systems," in *IEEE International Conference on Communications (ICC) Workshops*, 2021, pp. 1–6.

[19] T. Wink and Z. Nochta, "An approach for peer-to-peer federated learning," in *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2021, pp. 150–157.

[20] B. McMahan *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*. PMLR, 2017, pp. 1273–1282.

[21] W. Shi *et al.*, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[22] P. Voigt and A. Von dem Bussche, "The eu general data protection regulation (gdpr)," *A Practical Guide, 1st Ed., Cham: Springer International Publishing*, vol. 10, no. 3152676, pp. 10–5555, 2017.

[23] J. Hestness *et al.*, "Beyond human-level accuracy: Computational challenges in deep learning," in *Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming*, 2019, pp. 1–14.

[24] X. Lian *et al.*, "Can decentralized algorithms outperform centralized algorithms? A case study for decentralized parallel stochastic gradient descent," in *Advances in Neural Information Processing Systems*, 2017, pp. 5330–5340.

[25] A. Nilsson *et al.*, "A performance evaluation of federated learning algorithms," in *Proceedings of the Second Workshop on Distributed Infrastructures for Deep Learning*, 2018, pp. 1–8.

[26] S. A. Rahman *et al.*, "Internet of things intrusion detection: Centralized, on-device, or federated learning?" *IEEE Network*, vol. 34, no. 6, pp. 310–317, 2020.

[27] K. Chandiramani *et al.*, "Performance analysis of distributed and federated learning models on private data," *Procedia Computer Science*, vol. 165, pp. 349–355, 2019.

[28] I. Hegedűs *et al.*, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," *Journal of Parallel and Distributed Computing*, vol. 148, pp. 109–124, 2021.

[29] G. H. Lee and S.-Y. Shin, "Federated learning on clinical benchmark data: Performance assessment," *Journal of medical Internet research*, vol. 22, no. 10, p. e20891, 2020.

[30] S. Otoum *et al.*, "Blockchain-supported federated learning for trustworthy vehicular networks," in *IEEE GLOBECOM*, 2020, pp. 1–6.

[31] Warnat-Herresthal *et al.*, "Swarm learning for decentralized and confidential clinical machine learning," *Nature*, vol. 594, no. 7862, pp. 265–270, 2021.

[32] M. R. o. Akdeniz, "Millimeter wave channel modeling and cellular capacity evaluation," *IEEE journal on selected areas in communications*, vol. 32, no. 6, pp. 1164–1179, 2014.

[33] K. Gomez *et al.*, "Achilles and the tortoise: Power consumption in IEEE 802.11 n and IEEE 802.11 g networks," in *IEEE Online Conference on Green Communications (OnlineGreenComm)*, 2013, pp. 20–26.

[34] M. Rizwan and S. A. Abbas, "Median path loss, fading and coverage comparison at 3.5 ghz and 700mhz for mobile wimax," in *2008 IEEE International Multitopic Conference*. IEEE, 2008, pp. 266–271.

[35] M. J. Crisp *et al.*, "Uplink and downlink coverage improvements of 802.11 g signals using a distributed antenna network," *Journal of Lightwave Technology*, vol. 25, no. 11, pp. 3388–3395, 2007.

[36] A. Vishwanath *et al.*, "Energy consumption comparison of interactive cloud-based and local applications," *IEEE Journal on selected areas in communications*, vol. 33, no. 4, pp. 616–626, 2015.

[37] Y.-C. Chiu *et al.*, "Are we one hop away from a better internet?" in *Procs. of 2015 Internet Measurement Conference (IMC)*, pp. 523–529.

[38] S. Wang *et al.*, "Edge server placement in mobile edge computing," *Journal of Parallel and Distributed Computing*, vol. 127, pp. 160–168, 2019.

[39] M. Lenczner and A. G. Hoen, "CRAWDAD dataset ilesansfil/wifidog (ver. 2015-11-06)," Downloaded from: https://crawdad.org/ilesansfil/wifidog/20151106/session.

[40] H. H. Yang *et al.*, "Age-based scheduling policy for federated learning in mobile edge networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 8743–8747.

[41] Y. Yan *et al.*, "Risk minimization against transmission failures of federated learning in mobile edge networks," *IEEE Access*, vol. 8, pp. 98 205–98 217, 2020.

[42] J. Liu *et al.*, "Performance analysis and characterization of training deep learning models on mobile device," in *2019 25th IEEE International Conference on Parallel and Distributed Systems*, pp. 506–515.

[43] Y. Kochura *et al.*, "Batch size influence on performance of graphic and tensor processing units during training and inference phases," in *Intern'l Conf. on Computer Science, Engineering and Education Applications*. Springer, 2019, pp. 658–668.

[44] A. Nika *et al.*, "Energy and performance of smartphone radio bundling in outdoor environments," in *Proceedings of the 24th International Conference on World Wide Web*, 2015, pp. 809–819.

[45] N. P. Jouppi *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, 2017, pp. 1–12.

[46] T. Jakobs *et al.*, "Reducing the power consumption of matrix multiplications by vectorization," in *International Conference on Computational Science and Engineering (CSE)*. IEEE, 2016, pp. 213–220.

[47] E. S. Ali *et al.*, "Machine learning technologies for secure vehicular communication in internet of vehicles: recent advances and applications," *Security and Communication Networks*, vol. 2021, no. 1, p. 8868355, 2021.

[48] J. Luketina *et al.*, "Scalable gradient-based tuning of continuous regularization hyperparameters," in *International conference on machine learning*. PMLR, 2016, pp. 2952–2960.

[49] M. Courbariaux *et al.*, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Advances in neural information processing systems*, 2015, pp. 3123–3131.

[50] A. Ziller *et al.*, "Pysyft: A library for easy federated learning," in *Federated Learning Systems*. Springer, 2021, pp. 111–139.

[51] PyTorch Neural Network API. [Online]. Available: https://pytorch.org/docs/stable/nn.html

[52] G. Drainakis *et al.*, "Federated vs. centralized machine learning under privacy-elastic users: A comparative analysis," in *2020 IEEE 19th International Symposium on Network Computing and Applications (NCA)*, pp. 1–8.

[53] Q. Meng *et al.*, "Convergence analysis of distributed stochastic gradient descent with shuffling," *Neurocomputing*, vol. 337, pp. 46–57, 2019.

[54] C. R. Harris *et al.*, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[55] M. M. Bejani and M. Ghatee, "A systematic review on overfitting control in shallow and deep neural networks," *Artificial Intelligence Review*, pp. 1–48, 2021.

[56] Y. Chen *et al.*, "Asynchronous online federated learning for edge devices with non-iid data," in *2020 IEEE International Conference on Big Data (Big Data)*. IEEE, 2020, pp. 15–24.

[57] Samsung specifications: Eb-bg531bbe battery - 2400mah li-ion. [Online]. Available: https://batteriesglobal.com/products/samsung-eb\-bg531bbe-battery

[58] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–7.

[59] X. Zhang, M. Hong, S. Dhople, W. Yin, and Y. Liu, "Fedpd: A federated learning framework with adaptivity to non-iid data," *IEEE Transactions on Signal Processing*, vol. 69, pp. 6055–6070, 2021.

[60] F. E. Casado *et al.*, "Concept drift detection and adaptation for federated and continual learning," *Multimedia Tools and Applications*, pp. 1–23, 2022.

[61] M. R. Bachute and J. M. Subhedar, "Autonomous driving architectures: insights of machine learning and deep learning algorithms," *Machine Learning with Applications*, vol. 6, p. 100164, 2021.

[62] A. Alabbasi *et al.*, "On cascaded federated learning for multi-tier predictive models," in *IEEE ICC Workshops*, 2021, pp. 1–7.

[63] R. K. Nath and H. Thapliyal, "Machine learning-based anxiety detection in older adults using wristband sensors and context feature," *SN Computer Science*, vol. 2, no. 5, p. 359, 2021.

[64] G. I. Parisi *et al.*, "Continual lifelong learning with neural networks: A review," *Neural networks*, vol. 113, pp. 54–71, 2019.

[65] M. F. Criado *et al.*, "Non-iid data and continual learning processes in federated learning: A long road ahead," *Information Fusion*, vol. 88, pp. 263–280, 2022.

[66] F. Bayram *et al.*, "From concept drift to model degradation: An overview on performance-aware drift detectors," *Knowledge-Based Systems*, p. 108632, 2022.

[67] J. Lu *et al.*, "Learning under concept drift: A review," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, pp. 2346–2363, 2018.

[68] D. M. Manias *et al.*, "Concept drift detection in federated networked systems," in *IEEE Global Communications Conference*, 2021, pp. 1–6.

[69] G. Canonaco *et al.*, "Adaptive federated learning in presence of concept drift," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–7.

[70] "Making 5G proactive and predictive for the automotive industry," Industry White Paper, 5GAA Automotive Association, 2020.

[71] A. Palaios *et al.*, "Machine learning for QoS prediction in vehicular communication: Challenges and solution approaches," *IEEE Access*, 2023.

[72] X. Yin *et al.*, "A comprehensive survey of privacy-preserving federated learning: A taxonomy, review, and future directions," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–36, 2021.

[73] E. Jothimurugesan *et al.*, "Federated learning under distributed concept drift," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2023, pp. 5834–5853.

[74] N. Harth *et al.*, "Local & federated learning at the network edge for efficient predictive analytics," *Future Generation Computer Systems*, vol. 134, pp. 107–122, 2022.

[75] C. B. Mawuli *et al.*, "Semi-supervised federated learning on evolving data streams," *Information Sciences*, p. 119235, 2023.

[76] L. Gondara and K. Wang, "Pubsub-ml: A model streaming alternative to federated learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2, pp. 464–479, 2023.

[77] F. E. Casado *et al.*, "Ensemble and continual federated learning for classification tasks," *Machine Learning*, pp. 1–41, 2023.

[78] Y. Chen *et al.*, "Asynchronous federated learning for sensor data with concept drift," in *2021 IEEE Intl. Conf. on Big Data*, pp. 4822–4831.

[79] B. Ganguly and V. Aggarwal, "Online federated learning via non-stationary detection and adaptation amidst concept drift," *IEEE/ACM Transactions on Networking*, 2023.

[80] A. Paszke *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[81] J. Brownlee, *Deep learning for time series forecasting: predict the future with MLPs, CNNs and LSTMs in Python*. ML Mastery, 2018.

[82] Z. Charles *et al.*, "On large-cohort training for federated learning," *Advances in neural information processing systems*, vol. 34, pp. 20 461–20 475, 2021.

[83] J. Dessain, "Machine learning models predicting returns: Why most popular performance metrics are misleading and proposal for an efficient metric," *Expert Systems with Applications*, vol. 199, p. 116970, 2022.

[84] "5GAA Technical Report: ToD: System requirements analysis and architecture," https://5gaa.org/content/uploads/2021/09/5GAA_ToD_System_Requirements_Architecture_TR.pdf, accessed: 2023-01-28.

[85] G. Nardini *et al.*, "Simu5G–an OMNet++ library for end-to-end performance evaluation of 5g networks," *IEEE Access*, vol. 8, 2020.

[86] M. Series, "Guidelines for evaluation of radio interface technologies for imt-2020," *Report ITU*, vol. 2512, p. 0, 2017.

[87] "Cellmapper," www.cellmapper.net, accessed: 2023-01-28.

[88] "Openstreetmap," www.openstreetmap.org, accessed: 2023-01-23.

[89] "Worldstats," www.nationsencyclopedia.com/WorldStats/WDI-transport-vehicles.html, accessed: 2023-01-28.

[90] M. Behrisch *et al.*, "SUMO–simulation of urban mobility: an overview," in *Procs of the Third Interl. Conf. on Advances in System Simulation (SIMUL)*. ThinkMind, 2011.

[91] N. Yu *et al.*, "Minimizing energy cost by dynamic switching on/off base stations in cellular networks," *IEEE Transactions on Wireless Communications*, vol. 15, no. 11, pp. 7457–7469, 2016.

[92] "Infrastructure sharing: An overview (GSMA)," www.gsma.com/futurenetworks/wiki/infrastructure-sharing-an-overview, accessed: 2023-09-07.

[93] X. Liu *et al.*, "Key technologies for 5g co-construction and shared base station data automatic configuration," in *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2021, pp. 1559–1563.

[94] "Mobility report 2022," www.ericsson.com/4ae28d/assets/local/reports-papers/mobility-report/documents/2022/ericsson-mobility-report-november-2022.pdf, accessed: 2023-01-28.

[95] C. S. Evangeline *et al.*, "Safety and driver assistance in VANETs: an experimental approach for V2V," in *Int'l Conf. on Communication and Electronics Systems*, 2019, pp. 397–402.

[96] A. Narayanan *et al.*, "A variegated look at 5G in the wild: performance, power, and QoE implications," in *ACM SIGCOMM'21*, pp. 610–625.

[97] Y. Wang *et al.*, "Benchmarking the performance and energy efficiency of ai accelerators for ai training," in *20th IEEE/ACM Int'l Symposium on Cluster, Cloud and Internet Computing*, 2020, pp. 744–751.

[98] "CPU benchmarking," www.cpubenchmark.net/cpu.php?cpu=Intel+Xeon+Platinum+8168+%40+2.70GHz&id=3111, accessed: 2023-01-28.

[99] K. Fatahalian *et al.*, "Understanding the efficiency of GPU algorithms for matrix-matrix multiplication," in *ACM conf. on Graphics hardware*, 2004, pp. 133–137.

[100] R. Hernangomez *et al.*, "Berlin v2x," 2022. [Online]. Available: https://dx.doi.org/10.21227/8cj7-q373

[101] A. Reisizadeh *et al.*, "Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization," in *Int'l Conf. on Artificial Intelligence and Statistics*, 2020, pp. 2021–2031.

[102] H. Na *et al.*, "LSTM-based throughput prediction for lte networks," *ICT Express*, 2021.

[103] S. Mozaffari *et al.*, "Deep learning-based vehicle behavior prediction for autonomous driving applications: A review," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 1, pp. 33–47, 2020.

[104] "Predictive QoS and V2X service adaptation," Technical Report, 5GAA Automotive Association, 2022.

[105] P. Zhao *et al.*, "The case design of testing vehicle platooning in a vehicle-road cooperation system for a 5G cross-border scenario," in *IEEE 4th Int'l Conference on Civil Aviation Safety and Information Technology (ICCASIT)*. IEEE, 2022, pp. 135–139.

[106] D. Raca *et al.*, "Beyond throughput: A 4g lte dataset with channel and context metrics," in *Proceedings of the 9th ACM multimedia systems conference*, 2018, pp. 460–465.

[107] D. Xu *et al.*, "Understanding operational 5G: A first measurement study on its coverage, performance and energy consumption," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 479–494.

[108] S. Farthofer *et al.*, "An open mobile communications drive test data set and its use for machine learning," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 1688–1701, 2022.

[109] A. M. Mandalari *et al.*, "Experience: Implications of roaming in europe," in *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, 2018, pp. 179–189.

[110] T. Miekkala *et al.*, "Nordicdat: A cross-border predictive qos dataset," 2024. [Online]. Available: https://zenodo.org/records/10964584

[111] S. Schwarzmann *et al.*, "Ml-based qoe estimation in 5g networks using different regression techniques," *IEEE Transactions on Network and Service Management*, vol. 19, no. 3, pp. 3516–3532, 2022.

[112] S. Barmpounakis *et al.*, "LSTM-based QoS prediction for 5g-enabled connected and automated mobility applications," in *2021 IEEE 4th 5G World Forum (5GWF)*. IEEE, 2021, pp. 436–440.

[113] A. Palaios *et al.*, "Network under control: Multi-vehicle E2E measurements for ai-based qos prediction," in *IEEE 32nd Annual Int'l Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, 2021, pp. 1432–1438.

[114] R. Hernangómez *et al.*, "Berlin v2x: A machine learning dataset from multiple vehicles and radio access technologies," in *IEEE 97th Vehicular Technology Conference (VTC2023-Spring)*, 2023, pp. 1–5.

[115] D. Schäufele *et al.*, "Terminal-side data rate prediction for high-mobility users," in *IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*, 2021, pp. 1–5.

[116] A. Narayanan *et al.*, "A first look at commercial 5G performance on smartphones," in *Procs. of The Web Conference 2020*, pp. 894–905.

[117] A. Narayanan, E. Ramadan *et al.*, "Lumos5G: Mapping and predicting commercial mmwave 5G throughput," in *Proceedings of the ACM Internet Measurement Conference*, 2020, pp. 176–193.

[118] Y. Liu and C. Peng, "A close look at 5G in the wild: Unrealized potentials and implications," in *IEEE INFOCOM 2023*, pp. 1–10.

[119] K. Kousias *et al.*, "A large-scale dataset of 4g, nb-iot, and 5G non-standalone network measurements," *IEEE Comm. Magazine*, 2023.

[120] A. Narayanan *et al.*, "A comparative measurement study of commercial 5G mmwave deployments," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 800–809.

[121] R. A. Fezeu *et al.*, "An in-depth measurement analysis of 5G mmwave phy latency and its impact on end-to-end delay," in *Int'l Conf. on Passive and Active Network Measurement*. Springer, 2023, pp. 284–312.

[122] T. Tsourdinis *et al.*, "Ue network traffic time-series (applications, throughput, latency, cqi) in lte/5G networks," 2022. [Online]. Available: https://dx.doi.org/10.21227/4ars-fs38

[123] A. Safari Khatouni *et al.*, "An open dataset of operational mobile networks," in *Proceedings of the 18th ACM Symposium on Mobility Management and Wireless Access*, 2020, pp. 83–90.

[124] R. A. Fezeu *et al.*, "Roaming across the european union in the 5G era: Performance, challenges, and opportunities," in *IEEE International Conference on Computer Communications*, 2024.

[125] F. Burmeister *et al.*, "Measuring time-varying industrial radio channels for d2d communications on agvs," in *2021 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2021, pp. 1–7.

[126] R. Hernangómez *et al.*, "Towards an ai-enabled connected industry: Agv communication and sensor measurement datasets," *arXiv preprint arXiv:2301.03364*, 2022.

[127] V. Raida *et al.*, "Real world performance of lte downlink in a static dense urban scenario-an open dataset," in *2020 IEEE Global Communications Conference*, pp. 1–6.

[128] W. Lawrenz, *CAN System Engineering From Theory to Practical Applications*. Springer Science Business Media, 2013.

[129] R. B. Langley, "RTK GPS," in *GPS World*, 1998, pp. 70–76.

[130] "AT commands," https://wiki.teltonika-networks.com/view/AT_Commands, accessed: 2024-03-30.

[131] "Linux ifstat," https://man7.org/linux/man-pages/man8/ifstat.8.html, accessed: 2024-03-30.

[132] "Linux ping command," https://linux.die.net/man/8/ping, accessed: 2024-04-22.

[133] "Data distribution service specification version 1.4." "https://www.omg.org/spec/DDS/1.4/About-DDS", accessed: 2024-03-30.

[134] C.-X. o. Wang, "On the road to 6g: Visions, requirements, key technologies, and testbeds," *IEEE Communications Surveys & Tutorials*, vol. 25, no. 2, pp. 905–974, 2023.

[135] J. Sung and S.-j. Han, "Use of edge resources for dnn model maintenance in 5g iot networks," *Cluster Computing*, pp. 1–13, 2024.

[136] M. Antonini *et al.*, "Tiny-MLOps: A framework for orchestrating ml applications at the far edge of iot systems," in *IEEE international conference on evolving and adaptive intelligent systems*, 2022, pp. 1–8.

[137] S. Sekigawa *et al.*, "Web application-based webassembly container platform for extreme edge computing," in *GLOBECOM 2023-2023 IEEE Global Communications Conference*. IEEE, 2023, pp. 3609–3614.

[138] Z. Safavifar *et al.*, "Multi-objective deep reinforcement learning for efficient workload orchestration in extreme edge computing," *IEEE Access*, vol. 12, pp. 74 558–74 571, 2024.

[139] G. Baldoni *et al.*, "Managing the far-edge: Are today's centralized solutions a good fit?" *IEEE Consumer Electronics Magazine*, vol. 12, no. 3, pp. 51–61, 2021.

[140] I. Čilić *et al.*, "Towards service orchestration for the cloud-to-thing continuum," in *6th Int'l Conf. on Smart and Sustainable Technologies*. IEEE, 2021, pp. 01–07.

[141] T. o. Le-Anh, "An intelligent edge system for face mask recognition application," in *International Conference on Industrial Networks and Intelligent Systems*. Springer, 2022, pp. 107–124.

[142] N. T. Kien *et al.*, "Machine learning-based service function chain over uavs: Resource profiling and framework," in *31st Int'l Telecommunication Networks and Applications Conference*. IEEE, 2021, pp. 127–133.

[143] Canonical, "Microk8s," https://microk8s.io, November 2024, version 1.26.0.

[144] A. Angelovski and W. Güth, "When to stop—a cardinal secretary search experiment," *Journal of Mathematical Psychology*, vol. 98, p. 102425, 2020.

[145] N. Baganal-Krishna *et al.*, "A federated learning approach to qos forecasting in cellular vehicular communications: Approaches and empirical evidence," *Computer Networks*, vol. 242, p. 110239, 2024.

[146] 5G-IANA H2020 Project, "Use case 6: Network status monitoring," 2024, accessed: 2024-10-23. [Online]. Available: https://www.youtube.com/watch?v=ihKWqR34hWc

[147] C. King, "stress-ng: a tool to load and stress a computer system," 2014, version 0.13.08 or later. [Online]. Available: https://github.com/ColinIanKing/stress-ng