Energy-Aware Optimization and Machine Learning Frameworks for Sustainable Cognitive Networks

JUNIOR MOMO ZIAZET

A THESIS

IN

THE DEPARTMENT

OF

COMPUTER SCIENCE AND SOFTWARE ENGINEERING

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY (COMPUTER SCIENCE) AT

CONCORDIA UNIVERSITY

MONTRÉAL, QUÉBEC, CANADA

AUGUST 2025

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Mr. Junior Momo Ziazet

Entitled: Energy-Aware Optimization and Machine Learning Frameworks for Sustainable Cognitive Networks

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Computer Science)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

_____ Chair
Dr. Catherine Mulligan

_____ External Examiner
Dr. Guido Alberto Maier

_____ Arm's Length Examiner
Dr. Mirco Ravanelli

_____ Examiner
Dr. Tristan Glatard

_____ Examiner
Dr. Yann-Gaël Guéhéneuc

_____ Supervisor
Dr. Brigitte Jaumard

Approved by _____
Tse-Hsun (Peter) Chen
Graduate Program Director

August 11, 2025

_____
Mourad Debbabi, Dean
Gina Cody School of Engineering and Computer Science

Abstract

Energy-Aware Optimization and Machine Learning Frameworks for Sustainable Cognitive Networks

Junior Momo Ziazet, Ph.D.
Concordia University, 2025

This thesis presents a unified framework for enabling sustainable cognitive networks through the integration of machine learning and energy-aware optimization. As networks evolve toward 5G and beyond, growing demands in performance, energy efficiency, and autonomous management call for intelligent, scalable solutions. This work addresses these challenges through a holistic approach spanning four layers: data generation, infrastructure optimization, distributed learning, and predictive control.

To enable AI-driven intelligence, techniques are developed at the data layer that leverage data-centric AI to generate high-quality synthetic 5G packet-level data and refactor real-world urban data into machine learning-ready, 5G-like flow-level traces. These methods mitigate data scarcity and heterogeneity, providing realistic and diverse data essential for robust network learning systems.

In the infrastructure layer, the placement of disaggregated 5G components, including Distributed Units, Centralized Units, and User Plane Functions, is formulated as a large-scale optimization problem. The proposed decomposition-based and heuristic approaches improve energy efficiency by up to 14% while maintaining Quality of Service and responsiveness. Experiments in simulated 5G environments highlight the limitations of traditional peak-time-based planning.

For distributed learning, AFSL (Asynchronous Federated-Split Learning) and its energy-aware variant, AFSL+, are proposed to address client heterogeneity, achieving convergence time reductions of up to 13%. These frameworks selectively engage participants, reducing energy consumption by up to 55% without sacrificing accuracy and stability. To minimize communication overhead, Ada-AFSL is introduced, a dynamic compression technique that adapts to real-time bandwidth fluctuations. In realistic 5G and IoT scenarios, it achieves up to 82% data reduction while preserving performance and enhancing generalization.

Lastly, ST-SplitGNN and ST-SplitGNN+ are developed as spatio-temporal split learning models for accurate traffic prediction and uncertainty-aware resource allocation. Learning process is partitioned between local and centralized components: at the edge, temporal encoders capture node-specific traffic patterns, while a centralized Graph Neural Network with a learnable adjacency matrix models time-dependent inter-node dependencies. These models enable proactive scaling policies that align reliability with sustainability goals.

Together, these contributions form a cohesive architecture for sustainable cognitive networks. By uniting optimization with machine learning, this thesis supports the development of intelligent, efficient, and environmentally responsible communication systems for the 5G era and beyond.

Acknowledgments

First and foremost, I would like to express my deepest and most sincere gratitude to my advisor, Brigitte Jaumard, whose exceptional guidance, unwavering support, and genuine care have shaped every step of my Ph.D. journey. She not only taught me the fundamentals of conducting rigorous and meaningful research, but also showed me, by example, what it means to approach science with limitless curiosity, tireless dedication, and integrity. I am especially grateful for her patience and the compassion she showed during moments when I struggled, whether professionally or personally. It has been an honor and privilege to work alongside her over these years and to witness her extraordinary passion for research in action, a passion that is both inspiring and contagious. She gave me the freedom and encouragement to pursue my own ideas, the confidence to take risks, and the opportunity to share my work at numerous conferences, which enriched my academic and professional growth. Our many discussions, at times intense, at times lighthearted, have been invaluable, leaving a lasting mark not only on my career but also on how I think, question, and create.

I am also deeply grateful to my thesis committee members, in particular Guido Alberto Maier, Tristan Glatard, Yann-Gaël Guéhéneuc, Mirco Ravanelli, for generously dedicating their time, expertise, and insight to this work. Their constructive feedback, probing questions, and thoughtful suggestions consistently challenged me to think more critically and approach problems from new perspectives. Their engagement not only improved the quality and clarity of this thesis, but also broadened my understanding of the field in ways that will influence my research for years to come.

This work would not have been possible without the generous support of the funding sources that made my Ph.D. possible. I am grateful to the Natural Sciences and Engineering Research Council of Canada (NSERC) under project ALLRP 566589-21, and to InnovÉÉ (INNOV-R program) through a partnership with Ericsson-Global Artificial Intelligence Accelerator (AI-Hub) Canada in Montreal and Environment and Climate Change Canada (ECCC).

I wish to extend my sincere appreciation to Adel Larabi from Ericsson-Global Artificial Intelligence Accelerator AI-Hub Canada in Montreal, whose broad technical knowledge, generous support, and thoughtful guidance were instrumental in shaping this research. His ability to explain complex concepts in 5G networks with clarity and depth greatly expanded my understanding of the field. Our engaging discussions not only enhanced my technical expertise but also inspired me to explore new ideas with confidence. Special thanks also go to Pierre Thibault from the same team, who meticulously set up the technical environment and testbed required for developing and validating the frameworks proposed in this thesis. His dedication and readiness to assist ensured the smooth progress of this project, and without his invaluable contributions, this research would have been exceedingly difficult to realize. I am equally thankful to Konstatinos Vandikas and Selim Ickin from Ericsson Research, Sweden-Stockholm, for their insightful feedback, technical expertise, and constructive perspectives on my work in distributed and decentralized machine learning. Our meetings and collaborative work sessions were both challenging and enriching, equipping me with skills and knowledge that will undoubtedly serve me well in the next chapter of my career.

I consider myself truly fortunate to have been part of the Large-Scale Optimization Lab, surrounded by such an extraordinary group of people. The atmosphere fostered by my fellow lab mates was nothing short of exceptional, intellectually stimulating, supportive, and filled with genuine camaraderie. I am grateful to my lab mates and colleagues, many of whom have become close friends, for their encouragement, generosity, and the countless moments that made this journey both enriching and unforgettable. I would like to thank Leonardo Pereira Dias, Charles Boudreau, Sebastian Racedo Valbuena, Nguyen Phuc Tran, Quang Anh Nguyen and Oscar Humberto Delgado Collao for their friendship, insightful discussions, and readiness to share their expertise whenever I sought advice. Our long, often spontaneous conversations, sometimes about research, sometimes about completely random topics added a unique and joyful dimension to my Ph.D. experience. A special note of gratitude goes to Leonardo Pereira Dias, whose support and presence during some of my most challenging moments meant more than words can express. His rigorous approach to work, meticulous attention to detail, and high standards have left a lasting imprint on how I approach research and problem-solving.

To my family and friends, thank you for your constant encouragement, patience, and belief in me. Your support has been a reminder that there is a world beyond deadlines, and your laughter and company have kept me grounded through the highs and lows. A special tribute goes to my father, Victor Momo Ziazet, whose values, work ethic, and quiet strength have been a guiding light in my life. Your words of wisdom

and example have shaped who I am today, and I carry them with me in every step of this journey.

Finally, to my wife, Gabrielle Stella, thank you for believing in me, for reminding me that there is life outside the thesis, and for patiently listening to my "just five minutes" research updates that sometimes turned into hour-long tangents. This thesis is as much yours as it is mine. Your unwavering love, understanding, and encouragement have carried me through the hardest days and made the good ones even brighter. You have been my anchor when things felt uncertain, my cheerleader when progress seemed slow, and my safe haven through it all. Thank you for our many wonderful years and for always being on my side, even when a continent was between us. I could not have done this without you, and I am endlessly grateful to walk this journey of life with you by my side.

Contribution of Authors

The authors contributions of the manuscripts presented in this thesis are as follows: In each case, the candidate has played a leading role in the conception, design, implementation, and writing of the work.

- Chapter 3
  - **J.M. Ziazet**, C. Boudreau, B. Jaumard, O. Delgado, "Designing graph neural networks training data with limited samples and small network sizes".
    J.M. Ziazet: Conceptualization, Methodology, Implementation, Formal analysis, Data curation, Validation, Visualization, Writing – original draft. **C. Boudreau:** Investigation, Validation, Formal analysis, Visualization, Writing – original draft. **B. Jaumard:** Formal analysis, Validation, Writing – review & editing, Supervision, Project administration. **O. Delgado** Visualization, Validation, Writing – review & editing.
  - B. Jaumard and **J.M. Ziazet**, "5G E2E Network Slicing Predictable Traffic Generator".
    J.M. Ziazet: Conceptualization, Methodology, Implementation, Formal analysis, Investigation, Data curation, Validation, Visualization, Writing – original draft. **B. Jaumard:** Conceptualization, Formal analysis, Validation, Writing – review & editing, Supervision.
- Chapter 4
  - **J.M. Ziazet**, B. Jaumard, "Energy Efficient Placement of Logical Functionalities in 5G/B5G Networks".
    J.M. Ziazet: Conceptualization, Methodology, Implementation, Formal analysis, Investigation, Data curation, Validation, Visualization, Writing – original draft. **B. Jaumard:** Conceptualization, Formal analysis, Validation, Writing – review & editing, Supervision.
- Chapter 5

- R. A. Albuquerque, L.P. Dias, **J.M. Ziazet**, B. Jaumard, K. Vandikas, S. Ickin, C. Natalino, L. Wosinska, P. Monti, and E. Wong, "Asynchronous Federated Split Learning".

  J.M. Ziazet: Conceptualization, Methodology, Implementation, Formal analysis, Data curation, Validation, Visualization, Writing – original draft. **L.P. Dias** Investigation, Implementation improvement, Validation, Formal analysis, Writing – original draft. **R. A. Albuquerque** Investigation, Implementation improvement, Validation, Writing – review & editing. **B. Jaumard:** Formal analysis, Validation, Writing – review & editing, Supervision. **K. Vandikas:** Formal analysis, Validation, Writing – review & editing. **S. Ickin:** Validation. **C. Natalino:** Formal analysis, Writing – review & editing. **L. Wosinska** Validation, Supervision. **P. Monti** Supervision, Writing – review & editing. **E. Wong** Validation, Writing – review & editing.

- **J.M. Ziazet**, B. Jaumard, P. Thibault, "Enhancing Energy Management and Efficiency of Asynchronous Federated Split Learning".

  J.M. Ziazet: Conceptualization, Methodology, Implementation, Formal analysis, Investigation, Data curation, Validation, Visualization, Writing – original draft. **B. Jaumard:** Formal analysis, Validation, Writing – review & editing, Supervision. **P. Thibault:** Provided lab access, set up the testbed, and performed system maintenance.

- Chapter 6

  - **J.M. Ziazet**, L.P. Dias, B. Jaumard, P. Thibault, K. Vandikas and S. Ickin, "Real-Time Network-Aware Compression for Efficient Split Learning in Distributed Systems".

    J.M. Ziazet: Conceptualization, Methodology, Implementation, Formal analysis, Data curation, Validation, Visualization, Writing – original draft. **L.P. Dias** Investigation, Implementation improvement, Validation, Formal analysis, Writing – original draft. **B. Jaumard:** Formal analysis, Validation, Writing – review & editing, Supervision. **K. Vandikas:** Formal analysis, Validation, Writing – review & editing. **S. Ickin:** Validation. **P. Thibault:** Provided lab access, set up the testbed, and performed system maintenance.

- Chapter 7

  - **J.M. Ziazet**, B. Jaumard, "A Spatio-temporal Split Learning Framework for 5G and B5G Traffic Prediction".

J.M. Ziazet: Conceptualization, Methodology, Implementation, Formal analysis, Investigation, Data curation, Validation, Visualization, Writing – original draft. **B. Jaumard:** Formal analysis, Validation, Writing – review & editing, Supervision.

- **J.M. Ziazet**, B. Jaumard, ”Spatio-temporal Split Learning for Energy-Aware Proactive 5G/B5G Resource Management”.

  J.M. Ziazet: Conceptualization, Methodology, Implementation, Formal analysis, Investigation, Data curation, Validation, Visualization, Writing – original draft. **B. Jaumard:** Formal analysis, Validation, Writing – review & editing, Supervision.

Acronyms

3GPP 3rd Generation Partnership Project.

AFSL Asynchronous Federated Split Learning.

AI Artificial Intelligence.

ARES Adaptive Resource Aware Split Learning.

ASTGCN Attention Based Spatial-Temporal Graph Convolutional Network.

B5G 5G and Beyond.

BER Block Error Rate.

CAPEX Capital Expenditures.

CDF Cumulative Distribution Function.

CG Column Generation.

CN Core Network.

CNN Convolutional Neural Networks.

CP Control Plane.

CPU Central Processing Unit.

CU Centralized Unit.

DCRNN Diffusion Convolutional Recurrent Neural Network.

DML Distributed Machine Learning.

DN Data Network.

DRR Deficit Round Robin.

DU Distributed Unit.

E2E End to End.

eMBB enhanced Mobile Broadband.

ESCS Energy Saving Client Selection.

FIFO First In First Out.

FL Federated Learning.

FSL Federated Split Learning.

FW Firewall.

GAN Generative Adversarial Network.

GAT Graph Attention Network.

GCN Graph Convolutional Network.

gNB gNodeB.

GNN Graph Neural Network.

GoS Grade of Service.

GRU Gated Recurrent Unit.

ICT Information and Communication Technology.

IID Independent and Identically Distributed.

ILP Integer Linear Programming.

IoT Internet of Things.

IPR Intellectual Property Rights.

KPI Key Performance Indicator.

LP Linear Programming.

LQI Link Quality Indicator.

LSTM Long Short Term Memory.

MAC Medium Access Control.

MAE Mean Absolute Error.

MAPE Mean Absolute Percentage Error.

MEC Multi-Access Edge Computing.

MILP Mixed-Integer Linear Program.

MIoT Massive Internet of Things.

MIT Massachusetts Institute of Technology.

ML Machine Learning.

mMTC massive Machine-Type Communications.

NAT Network Address Translation.

NFV Network Function Virtualization.

NR New Radio.

NWDAF Network Data Analytics Function.

OPEX Operational Expenditures.

PDCP Packet Data Convergence Protocol.

PSL Parallel Split Learning.

QoS Quality of Service.

RAM Random Access Memory.

RAN Radio Access Network.

RLC Radio Link Control.

RMP Restricted Master Problem.

RMSE Root Mean Squared Error.

RPL Routing Protocol for Low-power and lossy networks.

RRC Radio Resource Control.

RSRP Reference Signal Received Power.

RSRQ Reference Signal Received Quality.

RSSI Received Signal Strength Indicator.

RTT Round Trip Time.

RU Radio Unit.

SDN Software-Defined Networking.

SFC Service Function Chain.

SFL Split Federated Learning.

SFLG Split Federated Learning Generalization.

SL Split Learning.

SLA Service Level Agreement.

SP Strict Priority.

ST-GCN Spatio-Temporal Graph Convolution Network.

ST-GNN Spatio-Temporal Graph Neural Network.

ST-SplitGNN Spatio Temporal Split Graph Neural Neural.

TCP Transmission Control Protocol.

TM Traffic Monitor.

TWAMP Two-Way Active Measurement Protocol.

UE User Equipment.

UPF User Plane Function.

URLLC Ultra-Reliable Low-Latency Communications.

VM Virtual Machine.

VNF Virtual Network Functions.

VoIP Voice over Internet Protocol.

WFQ Weighted Fair Queuing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Over the past decade, the number of connected devices and the volume of exchanged data in wireless networks have grown exponentially. With the advent of 5G and the emergence 6G on the horizon, this growth extends far beyond human-centric communication. A new wave of connected entities, ranging from sensors and industrial machines to autonomous vehicles and drones has emerged, fundamentally transforming the nature of network usage. According to Ericsson's Mobility Report [Ericsson, 2024], mobile data traffic is expected to triple between 2024 and 2030, surpassing 473 exabytes per month, while the number of Internet of Things (IoT) connections is projected to exceed 38 billion by 2030 [GSMA-Intelligence, 2023].

This shift in user profiles, from predominantly human-centric to machine-driven communication, has catalyzed the emergence of diverse and demanding use cases. Services such as cloud gaming, augmented and virtual reality, high-definition video streaming, and autonomous control systems require ultra-reliable, high-throughput, and low-latency connectivity. These evolving requirements place growing computational and energy burdens on both the radio access and core segments of the network.

However, this explosion in connectivity and data processing comes with significant energy and environmental costs. The Information and Communication Technology (ICT) sector currently accounts for 2–4% of global greenhouse gas emissions, with communication networks responsible for approximately 20–30% of that total [International Energy Agency, 2021]. Without targeted efficiency measures, 5G deployments, especially those involving infrastructure densification and edge computing could lead to a 160% increase in

energy consumption by 2030 compared to 2020 levels [InterDigital and Research, 2021]. Moreover, the integration of Artificial Intelligence (AI)/Machine Learning (ML) workloads at the network edge may further exacerbate energy demands, especially if not accompanied by efficient orchestration and adaptive resource management strategies.

To address the growing need for both service quality and sustainability, network architectures have undergone a fundamental transformation. Enabling technologies such as Software-Defined Networking (SDN), Network Function Virtualization (NFV), network slicing, Multi-Access Edge Computing (MEC), and advanced radio access (mmWave, THz) have redefined how networks are designed and operated [Yousaf et al., 2017; Benzekki et al., 2016]. These innovations introduce programmability, automation, and agility, paving the way for more intelligent and responsive network management. At the same time, AI and ML are being recognized as pivotal for realizing self-optimizing and self-organizing networks [Blanco et al., 2017].

In this evolving context, there is a critical need for intelligent, adaptive, and energy-efficient AI-driven solutions capable of handling the heterogeneity, dynamism, and scale of next-generation networks.

This thesis aims to contribute to this effort by proposing a unified, energy-aware AI architecture that integrates data-driven modeling, large-scale optimization, communication-efficient distributed learning, and predictive control. The objective is to enable cognitive and sustainable networks that can dynamically adapt to real-world constraints while minimizing their energy footprint.

1.2 Context

1.2.1 Enabling Technologies for Intelligent Networks

The transition towards intelligent, energy-efficient, and adaptive communication networks is underpinned by several foundational technologies. These technologies form the backbone of 5G and future beyond 5G (B5G) infrastructures, enabling programmability, flexibility, automation, and distributed intelligence.

Software Defined Networks

SDN is a networking paradigm designed to simplify and enhance network management by decoupling the control plane from the data plane [Kreutz et al., 2014; McKeown et al., 2008]. In conventional networks, each device, such as a switch or router independently handles both the control logic (deciding how to route

traffic) and the data forwarding. This tightly coupled architecture makes it difficult to implement network-wide policies or adapt dynamically to changing conditions.

SDN addresses this limitation by centralizing the control plane in a software-based controller, which manages the behavior of the underlying network devices via programmable interfaces (e.g., OpenFlow [McKeown et al., 2008]). The data plane remains distributed across the devices but operates under the control of the SDN controller. This separation enables a global view of the network, allowing operators to apply high-level policies, dynamically reconfigure routes, and optimize performance in real-time based on network conditions and objectives such as latency, bandwidth, or energy efficiency.

As a result, SDN offers a more agile, programmable, and scalable framework for managing complex networks. These characteristics make it particularly attractive for dynamic, multi-tenant environments like 5G and Beyond (B5G), where rapid service provisioning, quality-of-service guarantees, and energy-aware orchestration are critical.

Network Function Virtualization and Functional Disaggregation in 5G

NFV is a foundational technology in the evolution towards flexible, programmable, and service-oriented communication networks. It enables the decoupling of network functions from proprietary hardware appliances (e.g., firewalls, load balancers), allowing them to run as Virtual Network Functions (VNFs) on general-purpose computing infrastructure (e.g., server) [ETSI, 2014]. This abstraction reduces both Capital Expenditures (CAPEX) and Operational Expenditures (OPEX), while increasing scalability and resource utilization [Han et al., 2015]. NFV also enables on-demand deployment and elastic scaling of network functions, which is crucial for adapting to fluctuating traffic and service demands in real time. NFV is especially critical in 5G networks, where highly diverse and dynamic service requirements require a more modular and scalable architectural approach.

One of the most impactful applications of NFV in 5G lies in the functional disaggregation of both the Radio Access Network (RAN) and the core network [3GPP, 2018a]. Unlike the monolithic and hardware-centric systems of 4G, 5G adopts a highly disaggregated, software-defined architecture. A fundamental transformation in this regard is the functional split of the RAN. At the heart of the RAN is the gNodeB (gNB), the 5G equivalent of the base station in previous generations. The gNB serves as the access point that connects user devices (such as smartphones, sensors, or autonomous vehicles) to the 5G core network. It manages wireless communication over the air interface and handles critical functions like signal processing,

scheduling, and data routing.

In traditional RAN implementations, all gNB functions, including radio transmission and baseband processing were tightly integrated into a single physical proprietary hardware. In contrast, 5G introduces a virtualized and disaggregated gNB architecture, dividing it into three logical components that can be deployed and scaled independently on general-purpose computing platforms.

- Radio Unit (RU): The RU is responsible for analog and digital signal processing at the radio interface. It handles tasks such as digital beamforming, analog-to-digital conversion, and low-level physical layer functions. The RU is typically deployed close to or integrated with the antenna and connects to the DU via the fronthaul interface.

- Distributed Unit (DU): The DU executes real-time functions of the Medium Access Control (MAC), Radio Link Control (RLC), and parts of the physical layer. It is latency-sensitive and is often located near the RU to ensure strict timing requirements. DUs are suitable candidates for virtualization using edge cloud resources, enabling localized processing and faster responsiveness.

- Centralized Unit (CU): The CU handles non-real-time functions, including the Radio Resource Control (RRC) and Packet Data Convergence Protocol (PDCP). Because its processing is less latency-critical, the CU can be deployed in centralized data centers, supporting resource pooling and multi-cell coordination. The CU connects to the DU via the midhaul interface.

This architecture allows independent scaling and flexible deployment of each function, optimizing for latency, throughput, or energy efficiency depending on the scenario [Foukas et al., 2016].

On the core network side, NFV facilitates the virtualization of essential functions such as the User Plane Function (UPF), which is responsible for packet forwarding and traffic handling between the access and data networks. The UPF operates in the user data plane and is often distributed closer to the network edge to reduce latency and offload traffic from centralized cores through the so-called MEC technology [Hu et al., 2015; Taleb et al., 2017]. The strategic placement of these logical functions is crucial for enabling low-latency services and energy-efficient data routing in 5G network [Mijumbi et al., 2015].

Overall, NFV and functional disaggregation empower operators to build programmable networks that can be dynamically reconfigured based on service-level objectives and environmental conditions. They provide the architectural foundation for implementing intelligent control and optimization policies, including

those driven by machine learning, that optimize not only performance but also sustainability. By decoupling software from hardware, NFV not only supports faster innovation cycles but also paves the way for energy-aware orchestration, as resource allocation and placement can be optimized for both performance and sustainability. This thesis builds on these principles to propose new strategies for energy-aware function placement and orchestration under dynamic traffic conditions.

Network Slicing

As mentioned above, 5G introduces support for a wide variety of use cases, ranging from high-throughput broadband access to ultra-reliable low-latency control systems and large-scale IoT deployments. To accommodate this heterogeneity, the traditional "one-size-fits-all" architecture of legacy mobile networks (2G, 3G, and 4G) is no longer sufficient. Instead, 5G embraces *network slicing*, a paradigm enabled by SDN and NFV that allows multiple virtual networks, or *slices*, to be created and operated over a shared physical infrastructure [Zhang et al., 2017; ETSI, 2018; Foukas et al., 2017].

Each slice is logically isolated and customized to meet the specific Quality of Service (QoS), reliability, latency, and security requirements of a given application, tenant, or industry vertical. This allows operators to tailor network behavior to meet the divergent demands of modern services without requiring dedicated infrastructure.

The 3GPP and ITU-T standards define three primary service categories, each typically served by a dedicated slice [3GPP, 2018b]:

- **enhanced Mobile Broadband (eMBB)**: Focused on high data throughput and coverage for applications such as 4K/8K video streaming, virtual/augmented reality, and cloud gaming. eMBB slices require high capacity and moderate latency.
- **Ultra-Reliable Low-Latency Communications (URLLC)**: Designed for latency-critical and reliability-sensitive applications such as autonomous vehicles, industrial automation, and remote surgery. These slices must guarantee stringent delay bounds and near-perfect reliability.
- **massive Machine-Type Communications (mMTC)**: Targets massive-scale, low-power, and infrequent communication among a vast number of IoT devices (e.g., sensors, meters). mMTC slices prioritize scalability and energy efficiency over high bandwidth.

By enabling the coexistence of these diverse traffic types on a unified infrastructure, network slicing is central to the architectural vision of 5G and beyond. It not only facilitates efficient resource utilization but also enables service-level customization, isolation, and automation. In this thesis, we build upon this concept by proposing intelligent and energy-aware strategies for slice-specific resource allocation and traffic forecasting.

1.2.2 AI and Machine Learning in Networking

To support the diverse and dynamic service requirements of 5G and beyond (B5G) networks, traditional rule-based management approaches, often rigid and reactive are no longer sufficient. Instead, the integration of AI and ML into the network architecture has become not only beneficial but essential. AI enables networks to operate in a cognitive and autonomous manner by learning from vast volumes of real-time and historical data. This allows the system to predict traffic demand, detect anomalies, optimize resource allocation, and adapt policies dynamically without the need for manual intervention. Capabilities that are unattainable with static rule-based systems. For example, AI-powered functions can foresee congestion patterns and proactively trigger load balancing or network slicing adjustments to ensure QoS [Foukas et al., 2017]. Furthermore, AI integration helps reduce energy consumption by enabling predictive sleep modes for underutilized nodes and optimizing routing paths to minimize power usage [Zhang et al., 2017]. These intelligent mechanisms collectively transform the network into a self-optimizing and energy-efficient infrastructure, aligning with both performance and sustainability goals.

In recognition of AI's transformative potential, the 3rd Generation Partnership Project (3GPP), a standard-setting organization for mobile communication systems, has introduced the Network Data Analytics Function (NWDAF) into its technical specifications as a novel standard Control Plane (CP) function within the core network system. NWDAF is designed to collect diverse data generated from various distributed network functions (NFs) and employs AI algorithms for data analytics tasks, encompassing both training and inference processes [3GPP, 2022]. NWDAF supports critical applications such as load prediction, QoS estimation, user mobility analytics, and slice performance forecasting, thereby enabling informed and adaptive decision-making across the network.

As previously discussed, 5G network functions such as RUs, DUs, CUs, and UPFs are increasingly virtualized and distributed across MEC platforms and centralized core sites. In this distributed architecture,

centralized AI training introduces significant challenges, including high energy consumption due to large-scale data transfers and data privacy concerns in multi-tenant network environments. To overcome these limitations, operators are increasingly adopting distributed learning paradigms, such as federated learning and split learning [Ickin et al., 2022], which allow AI model training to occur closer to the data source. These approaches preserve data privacy, reduce communication overhead, and support cost-effective, scalable intelligence, making them well-suited for real-world 5G deployments [3GPP, 2024a,b].

1.3 Research Questions and Contributions

The integration of AI with the enabling technologies of 5G, such as NFV, SDN, MEC, and network slicing, paves the way for intelligent and autonomous network management. However, this transformation also exposes a series of complex technical and operational challenges across all layers of the network architecture: from data collection and infrastructure design to learning methodologies and real-time decision-making. In particular, multi-slice, multi-site networks operating under dynamic service demands and energy constraints require more than fragmented solutions; they call for a holistic and scalable framework that unifies data-driven intelligence with sustainable infrastructure management. To address this need, this thesis proposes a coherent five-part pipeline, spanning from realistic data generation, to energy-aware infrastructure optimization, to scalable distributed learning, and finally to predictive, graph-based control, and resource allocation. Each part addresses a specific challenge in the literature and contributes to building an end-to-end foundation for intelligent and sustainable network operation in 5G and beyond. Figure 1.1 illustrates the overall research workflow of the thesis and highlights the key research questions addressed at each stage.

1.3.1 Research Question 1: Lack of Realistic, Multi-Slice, Multi-Site Datasets

Challenges and Objectives

To ensure that AI models can effectively address real-world problems in B5G networks, access to realistic and comprehensive datasets is essential. These datasets form the empirical basis upon which AI algorithms learn to make informed decisions for optimizing network resource allocation, managing mobility, and ensuring QoS guarantees. However, the current landscape of publicly available datasets is limited in several key aspects. Many datasets are scarce, limited in scope, or unavailable due to privacy regulations and

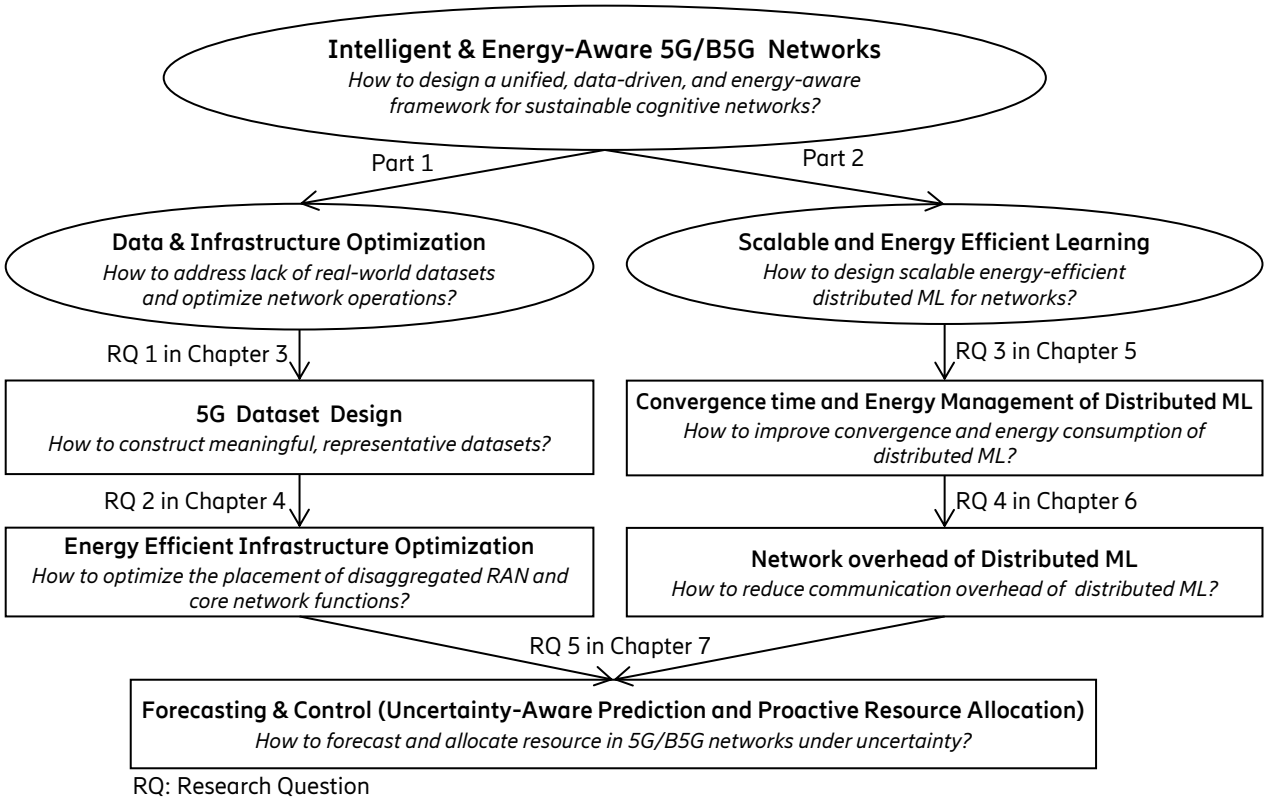


Figure 1.1: Overall research workflow: Illustration of the high-level research questions of the thesis, divided into two main research branches. The first branch focuses on enabling intelligent network operation through dataset generation and energy-aware infrastructure optimization. The second branch addresses the development of scalable and communication-efficient DML techniques for predictive control. Both branches converge towards the overarching objective of intelligent and sustainable network management in 5G and beyond (B5G) systems. Each block includes the associated research question that guides the investigation within that component.

proprietary restrictions. Furthermore, they often lack diversity across geographic sites, service slices, and traffic patterns, limiting their generalizability to practical network deployments [Taleb et al., 2017; Zhang et al., 2019a].

Although several 5G RAN traffic simulators have been proposed [Corcoran et al., 2020; Nardini et al., 2020; Patriciello et al., 2019], they are often tailored to specific use cases, lack generality, and do not support benchmarking across end-to-end provisioning scenarios. As a result, many ML-driven research efforts rely on either synthetic traffic [Guo et al., 2019a] or small-scale real traces [Cappanera et al., 2019; Chen et al., 2019b], which fail to capture the complex spatio-temporal dynamics, inter-slice dependencies, and variability inherent in real-world 5G networks.

This lack of high-quality, accessible data limits the development, validation, and benchmarking of intelligent algorithms, especially those that require detailed traffic behavior over time, space, and slices to operate effectively. It also impedes reproducibility and comparability of proposed methods, making it difficult to establish standardized baselines for evaluating progress in AI-native network management.

The primary objective of this part of the thesis is to address the dataset gap by generating and releasing high-fidelity, publicly available emulated datasets tailored for AI/ML research in 5G/B5G networks. These datasets will serve as a foundation for developing and evaluating learning-based algorithms under realistic multi-slice, multi-site, and dynamic traffic conditions.

Contributions

To overcome the limitations of existing datasets, this thesis develops and publicly releases a novel suite of high-fidelity emulated datasets that reflect the operational characteristics of realistic 5G networks. These datasets are generated using a custom-built emulation pipeline that integrates historical traffic trends, inter-slice dynamics, and multi-site deployments. Unlike existing tools and traces, the proposed datasets incorporate both temporal variation and geographic diversity across service slices (e.g., eMBB, URLLC, mMTC), enabling the training and robust evaluation of AI/ML models for intelligent network management. This dataset foundation plays a critical role throughout the thesis by supporting all subsequent components, including infrastructure optimization, energy-aware learning, and spatio-temporal predictive control. It also provides a reproducible platform for benchmarking future research in AI-native and sustainable 5G/B5G systems.

1.3.2 Research Question 2: Energy-Aware Optimization of Network Infrastructure

Challenges and Objectives

Building upon the data-centric foundation established in research question 1, the next key research challenge concerns the optimization of infrastructure deployment and operation in 5G and beyond (B5G) networks. While NFV and disaggregated RAN architectures introduce critical flexibility, they also complicate the energy-efficient and latency-aware placement of virtualized functions, namely RUs, DUs, CUs, and UPFs across edge and core sites.

On one hand, hosting logical functionalities closer to the RAN helps meet the ultra-low latency demands

of emerging use cases, such as URLLC, which require end-to-end latency as low as 1 ms to around 50 ms [Parvez et al., 2018; Ji et al., 2018]. Moreover, as global 5G traffic continues to surge, edge-proximal deployment becomes increasingly essential for maintaining service quality and responsiveness. On the other hand, this proximity introduces a significant energy burden, as edge cloud platforms that host these functions often offer lower resource consolidation efficiency, leading to higher cumulative power consumption [Jiang et al., 2019; Bianzino et al., 2010]. This presents a core trade-off: minimizing latency by deploying closer to users increases energy consumption, while consolidating resources centrally reduces energy costs but risks violating stringent latency requirements.

As such, jointly optimizing the placement of these network functions under energy, cost, and latency constraints becomes essential. This is a computationally challenging problem, due to the complex interplay between spatial traffic dynamics, resource capacities, and performance constraints [Bari et al., 2016]. While several studies have addressed individual aspects of this problem, such as DU and CU placement [Klinkowski, 2020; Liu et al., 2020; Yu et al., 2019] or UPF placement [Leyva-Pupo et al., 2019], the literature still lacks an integrated scalable optimization framework that holistically considers all these virtual functions in a multi-site, multi-slice architecture.

This infrastructure layer is not only responsible for ensuring the efficient operation of the network, but also forms the foundation upon which AI-driven control and prediction mechanisms can be effectively deployed. Without an optimized and resource-aware substrate, the learning models introduced in later stages would operate under suboptimal constraints, thereby limiting both network performance and the potential for energy savings.

The objective of this part of the thesis is to develop a holistic, scalable framework for energy-aware and latency-constrained optimization of network function placement in multi-site, multi-slice 5G/B5G architectures. This framework will enable efficient infrastructure provisioning that lays the groundwork for sustainable, AI-native network control.

Contributions

To address the challenges outlined above, this thesis introduces a novel mathematical optimization framework for the joint placement of DUs, CUs, and UPFs, along with the corresponding service request

routing. The framework is designed to account for spatial traffic dynamics, latency constraints, and heterogeneous infrastructure capabilities in multi-site and multi-slice networks. To tackle the computational complexity, a decomposition-based solution approach is proposed using column generation, which separates the problem into a master and pricing subproblem. This exact method is complemented by a scalable heuristic algorithm that supports rapid deployment decisions in larger-scale scenarios. The outcome is an energy and latency efficient deployment strategy that ensures optimized infrastructure provisioning while supporting real-time AI-based inference and control. This optimized infrastructure forms the physical substrate over which distributed intelligence developed in later stages of the thesis will operate, enabling end-to-end sustainability in next-generation network management.

1.3.3 Research Questions 3&4: Communication and Energy-Efficient Distributed Learning

Challenges and Objectives

Building on the optimized infrastructure from research question 2, where DUs, CUs, and UPFs are strategically deployed, the next layer of complexity arises in orchestrating the network at these locations. Enabled by components such as the NWDAF, AI models can be integrated into these distributed processing pools to optimize traffic management, mobility, and slice orchestration. However, the distributed and virtualized nature of modern 5G/B5G networks, combined with multi-tenancy, geographical dispersion, and stringent latency and privacy constraints, renders traditional centralized ML training impractical.

Centralized learning requires transferring large volumes of raw data to a central node for model training, which incurs significant communication overhead, increased energy consumption, and exposes sensitive user data to privacy and legal risks, including cross-border data transfer regulations and Intellectual Property Rights (IPR) concerns [Park et al., 2019; McMahan et al., 2017]. These challenges are amplified in multi-tenant deployments where data segregation and governance are paramount. These constraints motivate the shift towards distributed learning methods such as Federated Learning (FL) and Split Learning (SL), which train models collaboratively without sharing raw data, thereby preserving privacy and reducing backhaul traffic [Kairouz et al., 2021].

Nonetheless, distributed learning in the context of wireless networks introduces its own set of challenges. These include energy efficiency, non-Independent and Identically Distributed (IID) data across network sites [Lu et al., 2024; Zhao et al., 2021c], system heterogeneity [Li et al., 2020b], slow convergence under

straggling or disconnected nodes [Wu and Wang, 2021], and excessive communication overhead [Yang et al., 2022; Almanifi et al., 2023].

Existing solutions attempt to reduce communication cost using static compression techniques such as layer selection [Lin et al., 2024b; Li et al., 2024b; Samikwa et al., 2022], intermediate-data sampling [Chen et al., 2021a; Han et al., 2021; Wang et al., 2022a; Chopra et al., 2023], and quantization/sparsification [Oh et al., 2023; Zheng et al., 2023; Yeom et al., 2023; Yuan et al., 2020; Stich et al., 2018]. However, these methods often ignore dynamic network conditions, applying fixed compression levels throughout training, which may degrade accuracy under bad network conditions or overload networks during congestion.

Meanwhile, hybrid FL-SL frameworks have emerged as promising solutions to support non-IID data, device heterogeneity, and limited compute power in mobile/edge deployments. Yet, most of these methods adopt synchronous or weakly ε-asynchronous designs [Chen et al., 2021b; Zhou et al., 2022; Chai et al., 2021; Chen et al., 2020; Stephanie et al., 2023], which still suffers from suboptimal resource utilization and longer training time. Specifically, require all clients or cluster members to complete training before aggregation, lead to idle waiting and underutilized compute resources, and are not optimized for energy efficiency or real-time deployment in highly dynamic 5G/B5G environments.

In summary, while distributed learning methods such as FL and SL offer promising alternatives to centralized approaches, they remain constrained by synchronization bottlenecks, lack of adaptivity to network conditions, and insufficient support for heterogeneous, non-IID environments typical in real-world 5G/B5G deployments. These limitations hinder their scalability, energy efficiency, and applicability in dynamic, multi-tenant infrastructures.

The objective of this research is therefore to design a scalable, energy and communication efficient distributed learning framework that integrates the strengths of FL and SL in a fully asynchronous manner. This includes mechanisms for adaptive compression, heterogeneity-aware training, and dynamic client selection, enabling practical and sustainable AI-native control across the edge-to-core continuum of next-generation networks.

Contributions

To address the limitations of existing distributed learning frameworks in 5G/B5G networks, this thesis proposes novel frameworks that enables energy-efficient and communication-aware intelligence across distributed network nodes. Unlike existing FL or SL methods, which rely on global synchronization or static

compression, the proposed Asynchronous Federated Split Learning (AFSL) and its variants AFSL+ and Ada-AFSL introduce a fully asynchronous training mechanism that accommodates heterogeneous clients without incurring profiling overhead, and dynamically adjusts communication compression levels based on real-time bandwidth and latency conditions. The framework incorporates a novel label and entropy-based clustering method that mixes clients across clusters within each training round, promoting data diversity and improving model stability. It explicitly considers the interplay between client capabilities, data distribution, and network dynamics, ensuring scalable deployment in real-world, resource-constrained 5G/B5G environments. By eliminating synchronization bottlenecks and adapting to changing network conditions, the proposed frameworks advance the state of the art in enabling real-time, privacy-preserving, and energy-efficient distributed AI at the network edge.

1.3.4 Research Question 5: Distributed Spatio-Temporal Prediction and Intelligent Resource Control

Challenges and Objectives

Having addressed the foundational aspects of data availability (in research question 1), energy-efficient infrastructure deployment (in research question 2), and scalable distributed learning mechanisms (in research question 3 and 4), the final layer of complexity lies in leveraging these components for intelligent, energy-aware network control. Specifically, we focus on proactive resource allocation guided by accurate traffic prediction, a critical capability for dynamically adapting to load variations and minimizing energy waste in 5G and beyond (B5G) networks. In these systems, accurate traffic forecasting at fine spatio-temporal granularity is essential for enabling proactive resource provisioning, dynamically adjusting compute, bandwidth, and radio resources to current and anticipated demand. This capability is critical for minimizing energy waste through the deactivation of idle infrastructure components and for reducing over-provisioning without compromising service-level objectives [Zhang et al., 2019b; Chen et al., 2019a; Bianzino et al., 2010].

However, effective traffic prediction and control in multi-site, multi-slice networks is complicated by several key factors. First, traffic in operational networks exhibits complex and highly non-stationary spatio-temporal patterns, influenced by user mobility, service heterogeneity, and regional demand fluctuations [Wang et al., 2018, 2017, 2024]. Second, data privacy and regulatory restrictions often prevent raw traffic traces from being shared across sites or administrative domains, making centralized learning approaches

infeasible [Yu et al., 2020; Kougioumtzidis et al., 2025]. Third, many existing learning frameworks for traffic prediction either manage network sites independently or overlook inter-site correlations, leading to fragmented models that fail to generalize across heterogeneous network topologies [Aouedi et al., 2025].

Despite the increasing integration of machine learning-based traffic forecasting into network resource management, a critical gap persists in how these predictions are utilized for decision-making. Most existing resource allocation frameworks assume that the forecasted traffic values are accurate and deterministic, effectively treating point predictions as ground truth [Zhang et al., 2019a]. This assumption overlooks the inherent uncertainty in traffic dynamics, driven by user mobility, application diversity, and stochastic service demand, which can lead to suboptimal or even infeasible decisions when actual traffic deviates from the predicted values. In particular, underestimating demand may violate quality-of-service constraints, while overestimation leads to unnecessary energy expenditure. Thus, a more robust approach is needed, one that models and exposes uncertainty in forecasts and uses it to drive robust, risk-aware, and energy-efficient resource allocation.

The objective of this final part is to develop a distributed, uncertainty-aware spatio-temporal learning framework that jointly predicts traffic across multiple sites and slices, and uses these predictions to guide energy-efficient resource allocation under latency and QoS constraints. The proposed framework should (i) learn inter-site dependencies without requiring raw data sharing, (ii) expose uncertainty estimates for robust control, and (iii) operate under realistic communication and computation constraints imposed by distributed 5G/B5G environments. This part ultimately serves as the integration point for the thesis, bringing together the previously developed dataset, infrastructure, and learning capabilities into a unified, AI-driven network control that is both intelligent and sustainable.

Contributions

To address the outlined objectives, we develop a Spatio-Temporal Split Learning framework that integrates traffic prediction with uncertainty quantification and downstream resource optimization. The proposed architecture enables collaborative training across multiple sites and slices without centralized data aggregation, ensuring both scalability and data privacy compliance in distributed 5G/B5G deployments. At the client side, temporal models are used to capture slice-specific traffic dynamics, allowing each client to learn a representation tailored to its local temporal context and service characteristics. This client specialization is essential in multi-slice environments where traffic behavior varies widely across services (e.g.,

eMBB vs. URLLC), and ensures that learning remains sensitive to local variations in demand patterns.

On the server side, a Graph Neural Network (GNN) aggregates and refines these client representations, learning dynamic inter-site dependencies by constructing latent traffic graphs. This split learning architecture not only enhances predictive performance through collaborative representation learning but also enables flexible adaptation to heterogeneous network environments without requiring raw data sharing.

Importantly, our model goes beyond point forecasting by also predicting conditional quantiles of future traffic, thereby capturing uncertainty in load predictions. These uncertainty-aware forecasts are integrated into a robust resource allocation framework that minimizes energy consumption while satisfying latency and QoS constraints. By aligning forecasting with control decision windows and incorporating predictive uncertainty into the optimization process, the proposed solution closes the loop between learning and control.

This contribution completes the thesis by demonstrating how intelligent, distributed, and uncertainty-aware learning mechanisms can directly inform and optimize resource allocation decisions. It unifies the insights and tools developed in the earlier parts of the thesis, dataset generation, infrastructure optimization, and scalable learning, into a cohesive framework for energy-efficient, AI-enabled network control in 5G and beyond.

1.4 Thesis Organization

This thesis is organized into five core chapters, each addressing a critical aspect in building cognitive and sustainable communication networks. The progression of the chapters reflects a layered and integrated approach, beginning with data design and culminating in intelligent predictive control.

- **Chapter 2 – Preliminaries:** This chapter introduces the fundamental concepts and technical foundations used throughout the thesis. It covers key optimization techniques including Integer Linear Programming (ILP), column generation, and heuristics for solving large-scale deployment problems. It then presents core concepts in distributed machine learning, with emphasis on FL and SL for communication- and privacy-aware training in networked environments. Finally, it reviews the theory and application of GNNs for modeling spatial and relational dependencies in communication networks.

- **Chapter 3 – Traffic Data Generation for Intelligent 5G Network Management:** This chapter addresses the foundational problem of data availability for training machine learning models in networking. It presents two complementary strategies: (i) a simulation-based, packet-level data generation approach using data-centric AI principles; and (ii) a flow-level dataset refactoring methodology from real-world urban mobility traces. These contributions provide domain-specific, high-quality datasets that enable robust model training and evaluation.

- **Chapter 4 – Energy Efficient Placement of Logical Functionalities in 5G/B5G Networks:** This chapter formulates and solves the problem of placing logical 5G functions including DUs, CUs, and UPFs, under latency, capacity and energy constraints. The problem is modeled as a large- scale energy minimization task. A column generation-based decomposition approach is proposed, supported by a scalable heuristic. The solution achieves up to 14% energy savings, offering a significant improvement over static planning approaches.

- **Chapter 5 – An Asynchronous scheme for Convergence time and Energy Management in Distributed ML:** This chapter presents AFSL and AFSL+, two asynchronous federated-split learning protocols. AFSL integrates federated learning (FL) and split learning (SL) to mitigate straggler effects, support heterogeneous devices, and reduce convergence time. AFSL+ introduces a clustering-based client selection strategy that balances energy efficiency, data diversity, and model performance. These approaches enable sustainable distributed learning in mobile, dynamic, and resource-constrained environments.

- **Chapter 6 – Real-Time Network-Aware Compression for Efficient Split Learning in Distributed Systems:** This chapter addresses the challenge of communication overhead in distributed learning. It proposes a dynamic, bandwidth-aware compression scheme tailored for split learning. Unlike static methods, the approach adapts compression rates in real time to network conditions, reducing transmission volume by up to 82% while preserving or even improving model generalization. This contribution enhances the deployability of split learning in constrained and fluctuating environments.

- **Chapter 7 – Spatio-temporal Split Learning for Uncertainty-Aware Prediction and Proactive Resource Allocation in 5G/B5G Networks:** This chapter introduces Spatio Temporal Split Graph

Neural Neural (ST-SplitGNN) and ST-SplitGNN+, spatio-temporal split learning frameworks for traffic prediction and resource allocation. The models integrate temporal forecasting, graph-based spatial reasoning, and uncertainty-aware decision making. The result is improved prediction accuracy and adaptive, Service Level Agreement (SLA)-compliant provisioning strategies. ST-SplitGNN+ incorporates uncertainty estimates directly into resource control, pointing the way towards intelligent, closed-loop orchestration

- **Chapter 8 – Conclusion and Future Work:** The final chapter concludes the thesis by synthesizing the key contributions and articulating their impact on the future of network design. It reflects on the integration of data-driven modeling, optimization, distributed learning, and predictive control into a unified framework. The chapter also outlines future research directions, including continual learning, transformer-based control, runtime feedback loops, and end-to-end efficiency optimization across learning and control pipelines.

The chapters of this thesis present a coherent framework for building AI-enabled communication systems that are not only high-performing but also energy-aware, adaptable, and robust in the face of uncertainty.

1.5 List of Publications

The research work presented in this thesis has led to the following publications in peer-reviewed venues. These publications correspond to the main technical contributions described in Chapters 3 through 7 of the thesis. In each case, the candidate has played a leading role in the conception, design, implementation, and writing of the work. These works have been incorporated into the thesis with appropriate modifications for coherence and formatting.

International Journals

- **J.M. Ziazet**, C. Boudreau, B. Jaumard, O. Delgado, ”Designing graph neural networks training data with limited samples and small network sizes”. In *ITU Journal on Future and Evolving Technologies*, 2023, 4(3), pp. 492-502.

- **J.M. Ziazet**, B. Jaumard, ”Energy Efficient Placement of Logical Functionalities in 5G/B5G Networks” *[Submitted]*.

- **J.M. Ziazet**, L.P. Dias, B. Jaumard, P. Thibault, K. Vandikas and S. Ickin, "Real-Time Network-Aware Compression for Efficient Split Learning in Distributed Systems" *[Submitted]*.
- **J.M. Ziazet**, B. Jaumard, "Spatio-temporal Split Learning for Energy-Aware Proactive 5G/B5G Resource Management" *[Submitted]*.

International Conferences

- **J. M. Ziazet**, Jaumard, B., Duong, H., Khoshabi, P., Janulewicz, E, "A Dynamic Traffic Generator for Elastic 5G Network Slicing". In *IEEE International Symposium on Measurements & Networking (M&N)*, 2022, pp. 1-6.
- B. Jaumard and **J.M. Ziazet**, "5G E2E Network Slicing Predictable Traffic Generator", In *International Conference on Network and Service Management (CNSM)*, 2023, pp. 1-7.
- **J.M. Ziazet** and B. Jaumard, A. Larabi, N. Huin, "Placement of Logical Functionalities in 5G/B5G Networks". In *IEEE Future Networks World Forum (FNWF)*, 2023, pp. 1-7.
- R. A. Albuquerque, L.P. Dias, **J.M. Ziazet**, B. Jaumard, K. Vandikas, S. Ickin, C. Natalino, L. Wosinska, P. Monti, and E. Wong, "Asynchronous Federated Split Learning". In *IEEE International Conference on Fog and Edge Computing (ICFEC)*, 2024, pp. 1-7.
- **J.M. Ziazet**, B. Jaumard, P. Thibault, "Enhancing Energy Management and Efficiency of Asynchronous Federated Split Learning". In *IEEE International Conference on Communications (ICC) workshop*, 2025 *[Accepted for publication]*.
- **J.M. Ziazet**, B. Jaumard, "A Spatio-temporal Split Learning Framework for 5G and B5G Traffic Prediction". In *International Conference on Transparent Optical Networks (ICTON)*, 2025 *[Accepted for publication]*.

Patent

The following patent was filed in collaboration with Ericsson.

- **J. M. Ziazet**, L.P. Dias, B. Jaumard, K. Vandikas, S. Ickin, "Training and/or Use of a Split Machine Learning Model"

Chapter 2

Preliminaries

This chapter provides the foundational concepts and technical background necessary for understanding the proposed solutions developed in this thesis. It introduces optimization techiniques, distributed machine learning paradigms, and graph neural networks. These preliminaries set the stage for the methodologies proposed in the following chapters.

2.1 Optimization Techniques

In this section, we present fundamental optimization techniques underlying the routing and placement problems discussed throughout the thesis. Given the complexity and scale of modern network infrastructures, efficient decision-making often relies on formulating problems as mathematical programs. These programs can range from exact formulations, which are solved by classical optimization solvers, to heuristic methods required in large-scale or real-time settings. We focus on three key families of methods:

- Linear and Integer Linear Programming: Widely used for problems where constraints and objectives can be expressed in linear form. Integer Linear Programming (ILP) allows modeling of combinatorial decisions such as placement and routing.

- Column Generation (CG): A decomposition technique suited for large-scale ILPs. It enables solving a restricted version of the problem iteratively by generating configurations (columns) only as needed, often used in network design and scheduling.

- Heuristics and metaheuristics: Algorithms that provide feasible (though not always optimal) solutions

with lower computational cost, especially useful when dealing with NP-hard problems in dynamic environments.

Each of these methods plays a critical role in the Chapter 4 that follow, particularly in the design of energy-efficient placement strategies of network function.

2.1.1 Linear and Integer Linear Programming

Linear Programming (LP) is a foundational optimization technique used to determine the best possible outcome, such as maximizing profit or minimizing cost, in problems where both the objective function and constraints are linear. A standard LP involves optimizing a linear objective function subject to a set of linear inequality (or equality) constraints. Any LP problem can be reformulated in a canonical form, such as the following maximization model:

$$\begin{aligned} \text{Maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{Subject to} \quad & \sum_{j=1}^{n} a_{ij} x_j \leq b_i, \quad i = 1, \ldots, m, \\ & x_j \geq 0, \qquad\qquad j = 1, \ldots, n. \end{aligned} \tag{2.1}$$

Here, c_j, a_{ij}, and b_i are known coefficients, and x_j are the decision variables. Constraints of the form $\sum_j a_{ij} x_j \geq b_i$ can be converted by multiplying both sides by -1. Likewise, minimization problems can be transformed into maximization problems by negating the objective coefficients, i.e., minimizing $\sum_j c_j x_j$ is equivalent to maximizing $\sum_j (-c_j) x_j$.

Integer and Mixed-Integer Linear Programming

While LP assumes continuous decision variables, many real-world problems involve discrete decisions, such as whether to activate a network node or assign a user to a specific base station. ILP extends LP by requiring some or all variables to take integer (often binary) values. Such mathematical programming formulation are particularly suitable for combinatorial optimization tasks such as network function placement, discrete resource allocation, and routing.

When a problem includes both continuous and integer variables, it is referred to as a Mixed-Integer Linear Program (MILP). Despite their modeling capabilities, ILP and MILP problems are NP-hard and generally more computationally challenging than their LP counterparts.

Solution Techniques and Complexity

The field of LP began in 1947 with George Dantzig's development of the *simplex method* [Dantzig, 1948], a geometric approach that moves from vertex to vertex along the edges of the polytope (a compact convex set with a finite number of extreme points) to find the optimum. While simplex method is efficient in practice, it has exponential worst-case complexity.

In 1980, Khachiyan introduced the *ellipsoid method*, proving that LPs could be solved in polynomial time [Khachiyan, 1980], although the algorithm was mostly of theoretical interest. A major breakthrough came in 1984 with Karmarkar's *interior-point method*, which offered a practical polynomial-time algorithm for LPs [Karmarkar, 1984]. Today, LP solvers combine these techniques to achieve efficient and robust performance in large-scale applications.

Duality in Linear Programming

A key concept in LP theory is *duality*. Every LP (called the *primal*) has an associated *dual* problem, where the roles of constraints and variables are interchanged. If the primal is a maximization problem, the dual is a minimization problem, and vice versa. The primal and dual formulations of Model (2.1) are:

$$\begin{array}{llll} \text{Dual (left):} & \text{Minimize} & \displaystyle\sum_{i=1}^{m} b_i y_i & \\ & \text{Subject to} & \displaystyle\sum_{i=1}^{m} a_{ij} y_i \geq c_j & \\ & y_i \geq 0 & & \end{array} \qquad \begin{array}{llll} \text{Primal (right):} & \text{Maximize} & \displaystyle\sum_{j=1}^{n} c_j x_j & \\ & \text{Subject to} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \leq b_i & \\ & x_j \geq 0 & & \end{array} \tag{2.2}$$

Here, y_i are the dual variables corresponding to the primal constraints with $1 \leq i \leq m$ and $1 \leq j \leq n$.

LP duality is based on two central theorems [Dantzig, 2016]:

- The **Weak Duality Theorem** states that the objective value of any feasible dual solution is an upper bound (for maximization problems) on the primal objective.
- The **Strong Duality Theorem** asserts that if the primal has an optimal solution $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$, then the dual also has an optimal solution $y^* = (y_1^*, x_2^*, \ldots, y_m^*)$, and their objective values are equal:

$$\sum_{j=1}^{n} c_j x_j^* = \sum_{i=1}^{m} b_i y_i^*.$$

These duality properties are not only theoretically elegant but also practically useful. They form the basis for algorithms such as primal-dual methods and decomposition techniques like column generation, which are particularly valuable for solving large-scale MILPs.

2.1.2 Column Generation

ILP offers a powerful modeling framework for optimization problems. However, its computational complexity escalates rapidly with problem size, making it impractical for large-scale instances. Under certain conditions, though, ILPs can still be effectively addressed through advanced decomposition techniques such as *Column Generation* (CG).

Column Generation [Desaulniers et al., 2006] is a method tailored for solving large-scale LPs and ILPs that involve a vast number of variables, often too many to consider simultaneously. The technique is based on the insight that, in most cases, only a small subset of variables contributes to the optimal solution. Instead of solving the full problem directly, CG iteratively considers a restricted version containing a limited number of variables and dynamically adds new ones (called *columns*) only when they are promising.

Figure 2.1 illustrates the overall structure of the algorithm. The original problem is decomposed into two interconnected subproblems:

- **Restricted Master Problem (RMP)**: A restricted version of the original problem that includes only a subset of variables (columns).
- **Pricing Problem (PP)**: A subproblem that searches for new columns with the potential to improve the current RMP solution.

At each iteration, the RMP is solved to obtain dual variable values (denoted μ_i), which are then used by the PP to evaluate the solution improvement capability of new variables. This is done by computing the *reduced cost* of each candidate variable. The *reduced cost* of a variable represents the value by which the objective function may improve if the variable enters the solution.

Given the primal form in Model (2.2), the reduced cost of a variable x_j is defined as:

$$\bar{c}_j = c_j - \sum_i a_{ij} \cdot \mu_i \tag{2.3}$$

Here, c_j is the original cost coefficient of variable x_j, a_{ij} represents its contribution to constraint i, and μ_i

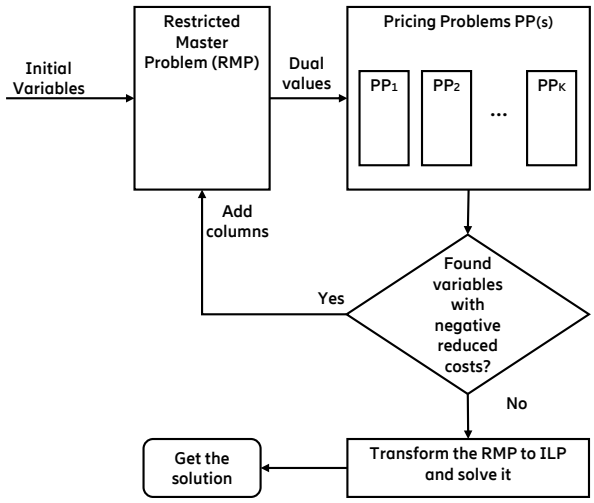


Figure 2.1: Flowchart of the column generation

is the dual value associated with that constraint. A variable with negative reduced cost (in minimization problems) signals an opportunity to improve the objective and is thus added to the RMP.

The algorithm alternates between solving the RMP and generating new variables via the PP, repeating this process until no more columns with negative reduced cost are found. At this point, the final RMP is solved as an ILP, yielding an integer solution based on the limited set of variables.

Although CG guarantees optimality for LPs, its application to ILPs requires additional care. Specifically, we first solve the linear relaxation of the ILP, i.e., the ILP without integrality constraints, using column generation. The resulting solution provides a lower bound z^*_{LP}. We then solve the final RMP as an ILP to obtain a feasible (and possibly optimal) integer solution with value z_{CG}. The quality of this integer solution can be assessed using the optimality gap (represented as follows for minimization):

$$\varepsilon = \frac{z_{CG} - z^*_{LP}}{z_{CG}}.$$

A value close to zero indicates that the solution is near-optimal; if the gap is exactly zero, z_{CG} is an optimal solution for the original ILP.

One of the key advantages of Column Generation is its ability to scale with modern hardware. Because each pricing problem can be solved independently, the method naturally benefits from parallel computing environments, particularly in systems with many core Central Processing Unit (CPU)s.

Nevertheless, CG is not without limitations. A major challenge is the phenomenon of *dual oscillation*,

which may hinder convergence when the number of constraints is large. This can be mitigated using stabilization techniques, such as those proposed in [Pessoa et al., 2018; Amor et al., 2004]. Furthermore, since CG solves the linear relaxation, additional procedures are necessary to derive optimal integer solutions. Well-established techniques for this purpose include Branch and Bound [Land and Doig, 2009], Branch and Cut [Hoffman and Padberg, 1985] and Branch and Price [Barnhart et al., 1998].

2.1.3 Heuristics

When dealing with large-scale, real-world optimization problems, such as network function placement, traffic engineering, or energy-efficient resource allocation, exact methods like ILP or Column Generation may become computationally prohibitive [Bertsimas and Tsitsiklis, 1997; Desrosiers and Lübbecke, 2005]. In such scenarios, heuristic algorithms provide practical alternatives, offering near-optimal solutions in significantly less time [Blum and Roli, 2003].

Heuristics are strategies that aim to produce sufficiently good solutions with reasonable computational effort, rather than guaranteeing global optimality. These methods typically rely on simplified rules or partial information to guide decision-making, trading off some accuracy for gains in speed or scalability. This makes them especially effective in time-sensitive or resource-constrained environments, such as communication networks, where rapid responses are critical [Michalewicz and Fogel, 2013].

Among the most widely used heuristic approaches are greedy algorithms [Cormen et al., 2022]. Greedy algorithms construct a solution incrementally by making the locally optimal choice at each step, aiming for immediate benefit. Despite their simplicity, greedy strategies often produce highly effective results, particularly for problems that exhibit the *greedy-choice property* or *submodularity* [Cormen et al., 2022; Nemhauser et al., 1978]. The *greedy-choice property* implies that a globally optimal solution can be constructed by choosing local optima at each step. *Submodularity*, on the other hand, refers to a diminishing returns property: the incremental gain from adding an element to a set decreases as the set grows. Many resource allocation and coverage problems in networks exhibit these properties, which allows greedy algorithms to perform near-optimally in such settings.

While greedy algorithms are not guaranteed to find the global optimum, their low computational complexity and ease of implementation make them attractive for large-scale network optimization. Moreover, their performance can be further enhanced using techniques such as local search, randomization, or integration into metaheuristic frameworks (e.g., genetic algorithms, simulated annealing) [Blum and Roli, 2003;

Talbi, 2009].

In Chapter 4, we demonstrate how greedy strategies can be effectively applied within our optimization framework to address the challenges of network-level decision-making.

2.2 Distributed Machine Learning

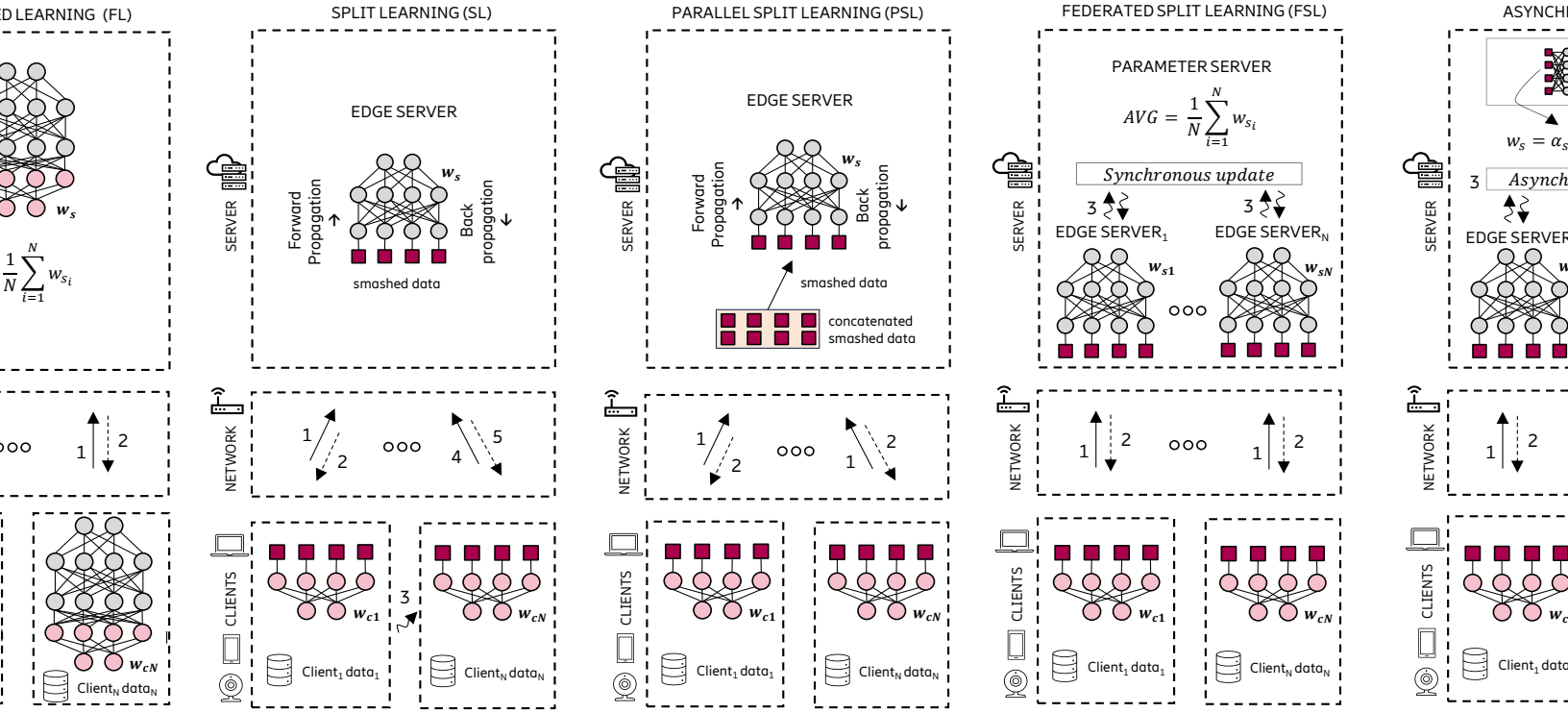
The proliferation of connected devices and the growing volume of data generated at the network edge have highlighted the limitations of traditional centralized machine learning approaches, particularly in terms of scalability, privacy, and communication overhead. Distributed learning has emerged as a powerful paradigm to address these challenges by enabling collaborative model training across multiple devices or nodes without requiring direct access to raw data [McMahan et al., 2017].

Unlike centralized learning, which aggregates all data at a central location for training, distributed learning strategies allow nodes to locally process and contribute to model training. This approach not only preserves data privacy but also reduces the burden on communication infrastructure, making it especially relevant for applications in wireless networks, IoT, and edge computing [Li et al., 2020b].

In this section, we discuss key distributed learning frameworks, including *Federated Learning*, which coordinates local model updates via a central server; *Split Learning*, which partitions models between clients and servers to reduce device-side computation; and hybrid or variant schemes that address specific system and performance constraints. These paradigms offer different trade-offs in terms of privacy, latency, bandwidth efficiency, and model accuracy, making them suitable for different classes of networked learning problems.

2.2.1 Federated Learning

Federated Learning [Bonawitz et al., 2019] is one of the most popular distributed learning techniques and has shown great potential in enabling a collaborative machine learning with the participation of hundreds or thousands of computation nodes (e.g., smartphones) each owning a portion of decentralized dataset. As depicted in Figure 2.2(a), in FL, each client in parallel trains (i.e., iterative forward and backward propagation) on the local dataset and updates their individual model parameters. Then, the clients periodically transmit the model weights to a server node (also known as the parameter server), in which the aggregation of all clients' weights takes place. The aggregated weights are then sent back to the clients. This way, the



(c) FSL representation

Figure 2.2: Representations of FL [Bonawitz et al., 2019], SL [Vepakomma et al., 2018], and FSL [Turina et al., 2021] architectures. In FL, each client possesses the full model, whereas in SL and FSL, the model is divided into client and server sub-models. Additionally, in FSL, each client maintains its edge-server copy. Training occurs concurrently across clients in FL and FSL, while in SL, training is conducted sequentially.

decentralized datasets are processed and used directly where they are collected, and collaboratively train a joint global model without sharing raw potentially sensitive user application datasets. One of the first use cases was next word prediction in Google applications running on Android phones [McMahan et al., 2017]. In the context of telecommunications, an early study on FL in the context of preventive maintenance was documented in [Brik et al., 2022; Singh and Nguyen, 2022] showcasing the reduction in the size of data that is transferred in a given link but also the possibility to allow for participants to join in the process late and without impacting overall model performance.

One advantage of FL is its scalability as multiple models are trained in parallel during each epoch [Turina et al., 2021]. However, as training an entire model is often computationally demanding, FL may not be suitable for resource-constrained devices such as smartphones and IoT devices [Duan et al., 2022], especially in scenarios involving larger models. Moreover, although data remains in the client during the entire training, the weights are shared and can be used to reconstruct the source dataset, e.g., inversion attacks, leading to a

potential data leakage [Turina et al., 2021].

2.2.2 Split Learning

Split learning [Vepakomma et al., 2018] is a method, first introduced in 2017 at Massachusetts Institute of Technology (MIT), which enables distributed learning between decentralized models in use cases when the decentralized models do not have the same input features and neural network architectures. As illustrated in Figure 2.2(b), it works by splitting a neural network into several parts that communicate by way of an interfacing, so called cut-layer. The layers of neural network models that are connected to each other are called cut-layers. This split property reduces the attack surface and privacy leakage as compared to FL and weight-transfer, since every participant only has access to subset of global observation. As such, in split learning there is one or more models that are trained locally on the data and the "smashed wisdom" of those models, which are in the form of encoded input data representations at their output, is then transmitted to the next model which uses that as input to proceed with the forward propagation step. In the same spirit, while doing backward propagation, the subset of gradients (or partial derivatives) of the cut-layer are communicated back to the first model to complete backward propagation while training. As such, this approach overcomes the need to transfer the entire model as opposed to federated learning. It is also better equipped to occupy constrained devices since smaller models can be associated with larger remote models that reside on data centers and enjoy more compute capability.

SL compared to FL usually has a higher communication cost when training small models with client nodes that have a large volume of data. Such cases are often found in IoT domain, where clients usually have constrained hardware, such as embedded systems. The high communication cost in split learning is due to the need to send out activations (during forward propagation), and to receive partial derivatives (during backward propagation), for every batch of training data and for every epoch. Furthermore, since the training process is sequential, the vanilla SL [Gupta and Raskar, 2018] does not consider computational heterogeneity, leading to computation resource under-utilization. In addition, as presented in [Turina et al., 2021], the model may not converge if the dataset in each client has unbalanced features, i.e., non-independent and identically distributed data (non-IID).

2.2.3 Federated Split Learning

Recently, to leverage the scalability of federated learning and the reduced computational requirements and stronger data privacy of split learning, hybrid approaches combining FL and SL have emerged in literature. One such approach is Federated Split Learning (FSL) [Turina et al., 2021]. In FSL, each client possesses an exclusive edge server located at the parameter server node. Figure 2.2(c) illustrates the FSL architecture and its functioning. Each client communicates intermediate data and gradients over the network with its respective edge server (1, 2). At the end of each epoch, all clients transmit their edge server's weights to a parameter server (3). Upon receiving all edge server weights, the parameter server aggregates them (e.g., averaging) (4) and redistributes them back to each edge server. This allows each client to perform forward and backward propagation in parallel, unlike traditional SL where these processes are sequential. While FSL has shown improvements in terms of training convergence time by enabling parallel client training, a limitation arises from the aggregation of FSL weights, which occurs only after all participating devices complete their respective training tasks. This aggregation process can be hindered by stragglers, potentially slowing down the overall training process. Additionaly, similar to SL, FSL suffers from high communication cost.

The methods defined in this section are foundational of our solutions proposed in Chapters 5, 6, and 7 .

2.3 Graph Neural Networks

Many problems in communication networks, such as traffic prediction, resource allocation, and user association, naturally exhibit a graph structure, where nodes represent entities (e.g., base stations, users, or network functions) and edges represent interactions or relationships (e.g., traffic flow, interference, or logical dependencies). Traditional neural networks fail to capture these relational inductive biases, which are critical in such settings. Graph Neural Networks (GNNs) have emerged as a powerful class of models capable of learning over graph-structured data by aggregating and transforming information from a node's local neighborhood [Wu et al., 2020a; Bronstein et al., 2017]. By leveraging the underlying topology, GNNs can model complex dependencies, support spatial generalization, and capture both local and global interactions, making them especially suitable for tasks in intelligent and adaptive network management. Recent advances in GNN architectures, including Graph Convolutional Network (GCN) [Kipf and Welling, 2016], Graph Attention Network (GAT) [Veličković et al., 2018], and spatio-temporal GNNs [Yu et al.,

2017], have demonstrated state-of-the-art performance in many domains and are increasingly applied to problems in wireless and distributed systems.

2.3.1 General Framework of Graph Neural Networks

GNNs operate on graph-structured data by learning node representations through an iterative process known as *message passing*. The core idea is that the hidden representation of a node u, $\mathbf{h}_u$ is computed based on its neighbors $N(u)$, allowing the model to capture both local topology and node features. At each iteration, nodes aggregate information from their neighbors and update their own representation. This process is repeated for a fixed number of iterations T, or until convergence, allowing information to propagate over progressively larger subgraphs. Figure 2.3 illustrates the message passing mechanism. The general message passing framework consists of the steps described below:

(1) **Initialization**: Node representations are initialized using their input features:

$$\mathbf{h}_u^{(0)} = \text{init}(\mathbf{x}_u), \quad u \in V. \tag{2.4}$$

(2) **Message Passing (Aggregation and Update)**: At each iteration $t = 1, 2, \ldots, T$, each node aggregates messages from its neighbors and updates its representation:

$$\mathbf{m}_{N(u)}^{(t)} = f^{(t)}\left(\{\mathbf{h}_v^{(t)} \mid v \in N(u)\}\right), \tag{2.5}$$

$$\mathbf{h}_u^{(t+1)} = g^{(t)}\left(\mathbf{h}_u^{(t)}, \mathbf{m}_{N(u)}^{(t)}\right). \tag{2.6}$$

(3) **Readout**: After T iterations, the final node embedding $\mathbf{z}_u = \mathbf{h}_u^{(T)}$ is used to perform downstream tasks:

$$\hat{\mathbf{o}}_u = r(\mathbf{z}_u), \quad u \in V. \tag{2.7}$$

(4) **Training**: For supervised learning, the model is trained by minimizing a loss over a labeled subset

$V_{\text{train}} \subset V$:

$$\mathcal{L} = \sum_{u \in V_{\text{train}}} \text{loss}(\hat{\mathbf{o}}_u, \mathbf{o}_u). \tag{2.8}$$

In this framework: init$(\cdot)$ initializes node embeddings from input features $\mathbf{x}_u$, often via zero-padding or

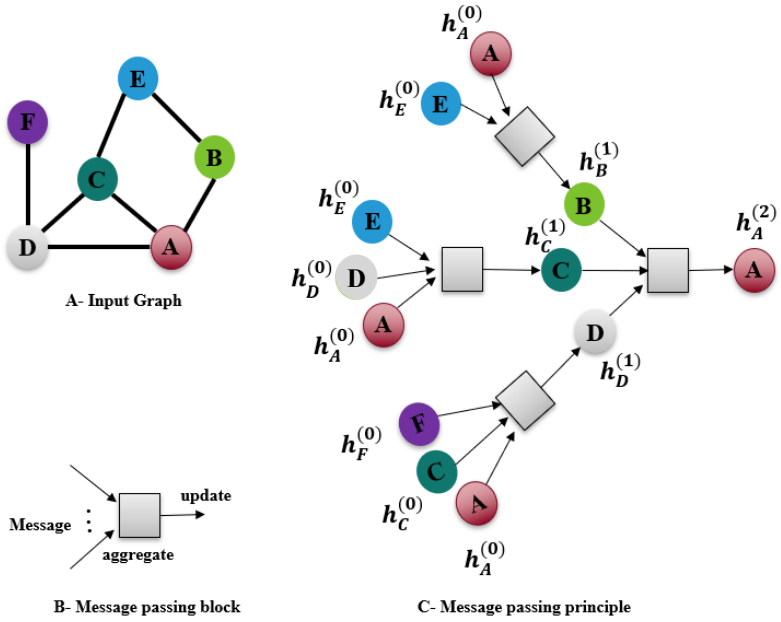


Figure 2.3: Illustration of message passing for learning node representations.

This message passing scheme enables each node to iteratively expand its receptive field, incorporating information from farther regions of the graph. As a result, each node representation can be interpreted as encoding a rooted subtree centered at that node. A key property of the aggregation function $f(\cdot)$ is that it must be *permutation-invariant*, meaning that it treats neighbors as an unordered set. Typical choices include element-wise summation, average, or maximum. The update and readout functions $g(\cdot)$ and $r(\cdot)$ are generally parameterized by neural networks such as multi-layer perceptrons (MLPs), enabling end-to-end differentiable training. In the next section, we present representative variants of GNNs. These models primarily

differ in how the aggregation and update functions are defined, with the goal of improving expressiveness and generalization performance.

2.3.2 The Basic Graph Neural Network

The aggregate and update functions of the basic GNN, a simplification of early models introduced by [Merkwirth and Lengauer, 2005] and [Scarselli et al., 2008], are defined as:

$$Aggregation : \mathbf{m}_{N(u)}^{(t)} = \sum_{v \in N(u)} \mathbf{h}_v^{(t)}, \tag{2.9}$$

$$Update : \mathbf{h}_u^{(t+1)} = \sigma\left(\mathbf{W}_{self}^{(t+1)} \mathbf{h}_u^{(t)} + \mathbf{W}_{neigh}^{(t+1)} \mathbf{m}_{N(u)}^{(t)}\right), \tag{2.10}$$

where $\mathbf{W}_{self}^{(t+1)}$ and $\mathbf{W}_{neigh}^{(t+1)}$ are trainable weight matrices and $\sigma(.)$ is a non-linear activation function (e.g., ReLU or tanh). A simplified version shares parameters across both matrices and adds self-loops, yielding the following formulation:

$$Update : \mathbf{H}^{(t+1)} = \sigma\left((\mathbf{A} + \mathbf{I})\mathbf{H}^{(t)} + \mathbf{W}^{(t+1)}\right), \tag{2.11}$$

where $\mathbf{A}$ is the adjacency matrix and $\mathbf{I}$ is the identity matrix.

2.3.3 Graph Convolutional Networks

Graph Convolutional Networks (GCNs), popularized by [Kipf and Welling, 2016], employ symmetric normalization of neighbor features and self-loops. The aggregation and update steps are given by:

$$Aggregation : \mathbf{m}_{N(u)}^{(t)} = \sum_{v \in N(u) \cup \{u\}} \frac{\mathbf{h}_v^{(t)}}{\sqrt{|N(u)||N(v)|}}, \tag{2.12}$$

$$Update : \mathbf{h}_u^{(t+1)} = \sigma\left(\mathbf{W}^{(t+1)} \mathbf{m}_{N(u)}^{(t)}\right). \tag{2.13}$$

Compared to the basic GNN formulation in Equation (2.9), the normalization used in GCN addresses a key limitation: the unnormalized sum of neighbor embeddings can lead to instability due to its sensitivity to node degrees. By incorporating degree-based normalization, GCNs provide more stable and consistent message aggregation across varying graph structures.

Beyond neighborhood normalization, several other strategies have been proposed to enhance GNN performance. For instance, [Zaheer et al., 2017] introduced *set pooling*, demonstrating that an aggregation function structured as in Equation (2.14) can serve as a universal approximator for permutation-invariant functions over sets. In contrast, *Janossy pooling* [Murphy et al., 2019] introduces permutation sensitivity by applying a function to ordered tuples of neighbors and averaging the results over multiple permutations, thus capturing richer structural dependencies in the graph.

$$Aggregation : \mathbf{m}_{N(u)}^{(t)} = MLP_\theta \left(\sum_{v \in N(u)} MLP_\phi(\mathbf{h}_v^{(t)}) \right). \tag{2.14}$$

MLP_θ and MLP_ϕ represent arbitrarily deep multi-layer perceptrons, each parameterized by their respective trainable parameters θ and ϕ.

2.3.4 Graph Attention Networks

In GCNs, the importance of a neighbor v to a target node u is determined by the weight of the edge $\mathbf{A}_{uv}$, which is normalized by the degrees of the connected nodes. However, in practice, input graphs can be noisy, and these predefined edge weights may not reliably capture the true strength of relationships between nodes. To address this limitation, a more robust approach is to learn the importance of each neighbor directly from the data. Graph Attention Networks (GAT), introduced by [Veličković et al., 2018], are based on this principle. They leverage the attention mechanism [Bahdanau et al., 2015] to dynamically compute attention scores for each neighbor during message passing. The aggregation function of GAT is defined in Equation (2.15).

$$Aggregation : \mathbf{m}_{N(u)}^{(t)} = \sum_{v \in N(u) \cup \{u\}} \alpha_{uv} \mathbf{h}_v^{(t)}, \tag{2.15}$$

where α_{uv} represents the attention coefficient assigned to neighbor $v \in N(u)$ during the aggregation of information at node u. These attention weights are computed as defined in Equation (2.16).

$$a_{uv} = \frac{exp(\mathbf{a}^T[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_v])}{\sum_{v' \in N(u)} exp(\mathbf{a}^T[\mathbf{W}\mathbf{h}_u \oplus \mathbf{W}\mathbf{h}_{v'}])}, \tag{2.16}$$

where **a** is a trainable attention vector, $\mathbf{W}$ is a trainable matrix, and $\oplus$ denotes the concatenation operation. Several variants of the attention mechanism have been proposed to enhance the expressiveness of GATs. For instance, bilinear attention is formulated in Equation (2.17), while attention mechanisms employing multi-layer perceptrons (MLPs) are illustrated in Equation (2.18):

$$a_{uv} = \frac{exp(\mathbf{h}_u^T \mathbf{W}\mathbf{h}_v)}{\sum_{v' \in N(u)} exp(\mathbf{h}_u^T \mathbf{W}\mathbf{h}_{v'})}, \tag{2.17}$$

$$a_{uv} = \frac{exp\left(MLP(\mathbf{h}_u, \mathbf{h}_v)\right)}{\sum_{v' \in N(u)} exp\left(MLP(\mathbf{h}_u, \mathbf{h}_{v'})\right)}. \tag{2.18}$$

Similarly, a multi-head attention mechanism inspired by the Transformer architecture [Vaswani et al., 2017] has been introduced, as shown in Equations (2.19) and (2.20).

$$Aggregation : \mathbf{m}_{N(u)}^{(t)} = [\mathbf{a}_1 \oplus \mathbf{a}_2 \oplus \cdots \oplus \mathbf{a}_K], \tag{2.19}$$

$$\mathbf{a}_k = \mathbf{W}_i \sum_{v \in N(u)} \alpha_{uvk} \mathbf{h}_v, \tag{2.20}$$

where the attention weights α_{uvk} for each of the K attention heads can be computed using any of the aforementioned attention mechanisms.

Incorporating attention is an effective strategy to enhance the representational capacity of GNN models, particularly in scenarios where the relative importance of neighboring nodes varies or is known to be significant.

2.3.5 Spatio-Temporal Graph Neural Networks

Spatio-Temporal Graph Neural Networks (ST-GNNs) extend GNNs to model temporal dynamics over structured data, such as traffic, mobility, or sensor networks. In this context, nodes have features that evolve over time, and dependencies exist both across the graph structure (spatial) and across time (temporal). A

general Spatio-Temporal Graph Neural Network (ST-GNN) layer integrates spatial message passing with temporal modeling. At iteration t, we define:

$$\text{Spatial Aggregation:} \quad \mathbf{m}_{N(u)}^{(t)} = f^{(t)}\left(\{\mathbf{h}_v^{(t)} : v \in N(u)\}\right), \tag{2.21}$$

$$\text{Temporal Update:} \quad \mathbf{h}_u^{(t+1)} = g^{(t)}\left(\mathbf{h}_u^{(t)}, \mathbf{m}_{N(u)}^{(t)}, \mathbf{h}_u^{(t-1)}, \dots\right). \tag{2.22}$$

The function $g^{(t)}$ is often instantiated using temporal models like Gated Recurrent Unit (GRU), Long Short Term Memory (LSTM), or 1D convolutions. Spatial aggregation $f^{(t)}$ typically reuses GCN, GAT, or similar layers. Several ST-GNN architectures exist: Diffusion Convolutional Recurrent Neural Network (DCRNN) [Li et al., 2017] combines diffusion convolution with GRU for traffic forecasting. Spatio-Temporal Graph Convolution Network (ST-GCN) [Yan et al., 2018] applies GCN over skeleton-based data for action recognition. Attention Based Spatial-Temporal Graph Convolutional Network (ASTGCN) [Guo et al., 2019b] uses spatial and temporal attention for better interpretability and performance.

ST-GNNs are particularly well-suited for spatio-temporal prediction tasks where dependencies are both topological and temporal.

These models form the foundational building blocks for the data-centric AI approach proposed in Chapter 3, and the predictive learning framework developed in Chapter 7.

Chapter 3

Traffic Data Generation for Intelligent 5G Network Management

This chapter focuses on data generation strategies tailored to network traffic in 5G and beyond systems. It explores two distinct but complementary use cases, each motivated by different assumptions regarding data availability and modeling objectives. The chapter develops methodologies aimed at generating high-quality traffic data to support machine learning applications in networking.

3.1 Introduction

Data fuels the training and validation of Machine Learning (ML) models responsible for forecasting traffic demands, allocating resources, and making real-time decisions across diverse network slices and infrastructures. However, despite the proliferation of research in this domain, the availability of representative, large-scale 5G traffic datasets remains severely limited. Real-world traffic datasets often suffer from various limitations, including scarcity, limited coverage, and privacy restrictions, which impede their open accessibility and reproducibility [Taleb et al., 2017; Zhang et al., 2019a].

In the field of machine learning for networking, researchers typically rely on one of two types of data representations: packet-level data or flow-level data. Each has distinct characteristics, advantages, and limitations, leading to different modeling assumptions and research strategies.

Packet-level datasets offer fine-grained visibility into network activity, capturing per-packet metadata such as timestamps, source and destination IPs, port numbers, protocols, and payload sizes. These datasets

are instrumental in tasks like intrusion detection [Tavallaee et al., 2009], traffic classification [Aceto et al., 2019], and Key Performance Indicator (KPI) prediction [Tran et al., 2023]. However, their use is often limited in practice due to challenges such as data privacy concerns, data sensitivity and high costs of data collection and storage. Furthermore, access to such data typically requires deep integration into operator networks, which is infeasible for many researchers. In cases where real packet-level data is unavailable, synthetic data generation becomes a valuable alternative. Simulation tools such as OMNeT++ [Varga and Hornig, 2008] allow researchers to model and generate realistic traffic traces under controlled conditions. Unfortunately, generating a large amount of samples with these simulators is often prohibitively costly. In this context, the first part of this chapter focuses on a packet-based use case, where we leverage data-centric AI principles [Andrew et al., 2021; Zha et al., 2025] to produce high-quality synthetic datasets tailored to specific ML tasks. In particular, we concentrate on KPI prediction to support intelligent network management.

On the other hand, flow-level datasets abstract away individual packet-specific details and instead aggregate statistics over communication sessions or time intervals. Common features include flow duration, or/and total byte counts. These datasets are more readily available in practice, particularly in operator networks where scalable traffic monitoring solutions rely on flow-level telemetry. Flow-level data supports use cases such as network-wide traffic forecasting [Zhang et al., 2019a], capacity planning, and anomaly detection [Pang et al., 2021]. Here, the assumption is that similar historical data exists and can be leveraged for new predictive tasks. As such, the second part of this chapter addresses a flow-based use case, where real-world open datasets, specifically, urban mobility traces, are transformed and refactored into structured network traffic datasets. These refactored datasets retain the temporal and spatial characteristics of real-world environments, providing a meaningful testbed for evaluating ML models under noisy and dynamic conditions.

By addressing both packet-level and flow-level scenarios, this chapter responds to the practical diversity of data access in networking research. It offers complementary strategies, simulation-based synthetic data generation and real-world data refactoring to support effective ML model development regardless of data scarcity or noise. This dual approach is critical for the broader adoption and reliability of AI-driven techniques in next-generation communication networks.

3.2 Related Work

This section reviews the literature relevant to data generation for 5G traffic modeling and learning under data scarcity. We categorize related efforts into (i) simulation-based and generative approaches for traffic data generation, and (ii) machine learning techniques designed to operate effectively with limited or heterogeneous data, with an emphasis on graph-based models.

3.2.1 Simulation-based and Generative Approaches for 5G Traffic Generation

The scarcity of large-scale, open, and temporally rich 5G traffic datasets has prompted researchers to explore simulation-based and generative approaches for traffic data generation. A common strategy involves using network simulators, such as OMNeT++ [Varga and Hornig, 2008] and its 5G extension Simu5G [Nardini et al., 2020], to emulate the behavior of 5G New Radio (NR) networks. These tools support the end-to-end modeling of the data and control planes, offering visibility into protocol-level behavior and enabling performance evaluation under controlled conditions [Deinlein et al., 2020]. However, despite their protocol accuracy, simulators like Simu5G lack realistic, pattern-driven traffic generators. Furthermore, generating a large amount of samples with these simulators is often prohibitively costly. This makes them unsuitable for training ML models on time-varying traffic demand, which requires predictable and repeatable traffic patterns.

To address the limitations of simulators, some researchers have turned to trace-based methods. For instance, [Raca et al., 2020] introduced a 5G dataset collected from a major Irish operator using G-NetTrack Pro. While the dataset includes rich KPI-level data (e.g., throughput, signal quality), the traffic load was artificially driven by fixed mobility and application profiles (e.g., video streaming), limiting its value for organic traffic prediction. Building on this dataset, [Kim et al., 2022] proposed a Generative Adversarial Network (GAN)-based neural traffic generator for synthesizing traffic patterns in private 5G deployments. Similarly, [Corcoran et al., 2020] proposed a parameter-sampling approach to generate statistically consistent traffic for simulations. Although these methods produce useful training data for control and resource management tasks, they fall short for forecasting applications, where temporal regularity and historical continuity are essential.

In recent years, data-centric AI has emerged as a paradigm shift, emphasizing data quality over model complexity [Andrew et al., 2021; Zha et al., 2025]. In this context, generative models such as GANs can

be used not to generate final traffic traces, but to synthesize intermediate inputs (e.g., topologies, routing policies, traffic matrices) that are then passed through simulators. However, as noted in [Zhao et al., 2020], GANs require substantial training data to generalize well, and training them with limited samples often leads to mode collapse. Furthermore, topological realism in communication networks imposes additional challenges for graph-based data generation, requiring structure-preserving augmentation strategies [Zhao et al., 2021b].

3.2.2 Machine Learning under Data Scarcity and Structure Constraints

When operating under limited or heterogeneous data, especially in networking contexts where access to real traffic traces is constrained, machine learning systems must be adapted to generalize across domains and data regimes. This is particularly true for graph-based learning models used in traffic prediction, which must often handle varying graph sizes, noise levels, and temporal dynamics.

Several strategies have been proposed to improve model robustness and generalization in these settings. Data augmentation for graphs, as explored in [Zhao et al., 2021b], modifies graph structures (e.g., by adding or removing edges) or perturbs node features to simulate new samples. Other methods apply feature masking or propagation models to create diverse input scenarios [Feng et al., 2020]. While effective in large-data regimes, these approaches have limited impact when the training data is inherently sparse or structurally dissimilar from the test distribution. Alternative methods include data filtering, where out-of-distribution samples are excluded to improve train-test consistency [Ilyas and Rekatsinas, 2022], and early stopping, which prevents overfitting to noisy data during training [Li et al., 2020a]. In settings where data cannot be expanded, such as simulation-constrained environments, the focus shifts from quantity to quality, optimizing the informativeness and relevance of each training sample becomes paramount.

The literature shows that while simulation tools and generative models can help compensate for the scarcity of real 5G traffic data, they often lack temporal realism or require extensive training data which is costly to generate. Data-centric techniques, especially in combination with spatio-temporal and graph-based ML models, offer new possibilities but face unique challenges under limited data availability. This chapter builds on these foundations by proposing hybrid strategies. combining simulation, refactoring, and data-centric ML, to create usable traffic datasets under both synthetic and real-world constraints.

3.3 Use case 1: Packet-based Traffic Data Generation via Simulation

In this section, we adopt data-centric AI principles to generate synthetic packet-level datasets. These datasets are crafted through simulation to ensure they meet the statistical and structural requirements of the target ML tasks, offering a controlled and customizable foundation for model development and evaluation.

3.3.1 System Model and Problem Formulation

Preliminaries: RouteNet-Fermi

The RouteNet-Fermi model [Ferriol-Galmés et al., 2023a] is a state-of-the-art graph neural network-based model for network simulation. The model goes beyond simply reproducing the network topology as graph data, instead creating an heterogeneous graph that models the interactions between the traffic flows, queues and links of a network. A flow's state depends on the states of links and queues it encounters, a queue's state is influenced by the states of flows that pass through it, and a link's state is influenced by the state of queues that may lead network traffic into it. These circular dependencies are addressed through a three-part, custom message-passing scheme, shown in Figure 3.1. After the states of each of the flows, queues and links are initialized with the initial features, message-passing occurs iteratively, with the queue states being updated first, using aggregated states of the flows encountered by each queue, followed by the link states, using the states of queues that feed into the links, and flow states, using information aggregated from the states of all links and queues that compose a flow. This process repeats for T iterations, where T is a user-defined parameter before the flow states are finally used to compute the performance metric estimations. Furthermore, in a manner similar to the model in [Ziazet et al., 2022b], RouteNet-Fermi also features design choices such as replacing the numerical "link capacity" value with a relative value representing the traffic load of a link based on its capacity, and the delay is inferred from the queue occupancy rather than predicted directly, which helps retain accuracy even when testing on networks much larger than those experienced during training.

Problem formulation

The goal is to investigate data-centric strategies for enhancing the performance of Graph Neural Network (GNN) based models in network simulation tasks, under strict constraints on data availability and model flexibility. The primary objective is to generate a compact but high-quality training dataset that enables a

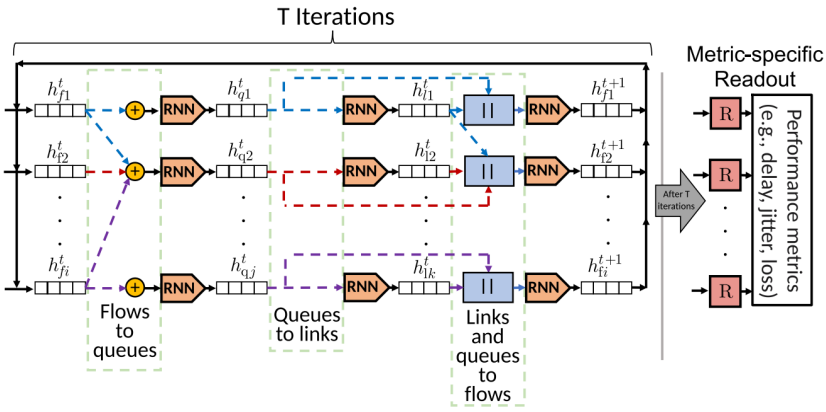


Figure 3.1: Schematic representation of Routenet-Fermi's message passing [Ferriol-Galmés et al., 2023a]

fixed GNN model to generalize effectively to unseen and more complex network scenarios. In this context, the problem is framed as a data-centric learning task where only the data, rather than the model architecture or training procedure can be modified.

We consider the GNN model, data assumptions, and constraints provided in an international benchmark setting, namely the third edition of the Graph Neural Networking Challenge [gnnet2022, 2022], which promotes the exploration of data-centric AI approaches for constructing accurate network digital twins. This setup is adopted in our study because it offers a standardized and rigorous framework in which our proposed solution can be directly compared with those of other international research teams. Such a comparative evaluation enables an objective assessment of the quality and generalizability of our approach. Specifically, we adopt RouteNet-Fermi [Ferriol-Galmés et al., 2023a], a state-of-the-art GNN model designed to predict per-flow performance metrics (e.g., delay) based on network state and configuration. The model takes as input a tuple consisting of: A topology object that describes the nodes and links of a network; A routing matrix specifying the path used between each pair of nodes; A traffic matrix that characterizes traffic flows between source-destination pairs. The label associated with each sample is the average delay per flow, which is produced through a packet-level simulation using a custom OMNeT++-based simulator. These labels are not analytically derived but generated through discrete-event simulation, emphasizing the realism and complexity of the problem setting.

Assumptions and Constraints

This study operates under several practical constraints, which are representative of real-world scenarios where large-scale, labeled training datasets are difficult or expensive to obtain:

- **Fixed model and training pipeline:** The GNN model architecture (RouteNet-Fermi), learning rate (0.005), optimizer (Adam), loss function (mean absolute percentage error), number of epochs (maximum of 20), and random seed are all fixed. No modifications to the model design, training procedure, or hyperparameters are permitted.
- Data generation constraints:
  - A maximum of 100 training samples may be generated and used.
  - Each training sample must be created using the provided packet-level simulator.
  - Topology constraints:
    - Topologies may include up to 10 nodes.
    - All links must be bidirectional, with bandwidths between 10,000 and 400,000 bits, in steps of 1,000.
  - Node configurations:
    - Each node must be assigned one of four predefined scheduling policies.
    - Buffer sizes must be between 8,000 and 64,000 bits.
  - Routing:
    - Each source-destination pair must have a single routing path, which may differ in the reverse direction.
  - Traffic flow characteristics
    - Each source-destination pair is associated with one traffic flow.
    - Average bandwidths of flows must be between 10 and 10,000 bps.
    - Flow sizes must range from 256 to 2,000 bits, with user-defined probability distributions summing to 1.
    - Flows may follow one of three predefined time distributions.

    - Each flow is assigned one of three types of service (ToS), which interact with node scheduling policies.

- Training and generalization gap: The training set must be generated using small network topologies ($\leq$ 10 nodes), while the validation and test scenarios involve larger, more complex topologies that are not directly accessible during training. This setup reflects the real-world challenge of training on limited, small-scale data while deploying in production-scale environments.

Objective

The core objective is to design and implement a data generation pipeline that can synthesize a compact yet diverse set of training samples satisfying all aforementioned constraints. The generated dataset must enable the fixed GNN model to generalize to unseen, large-scale topologies and accurately predict per-flow delay. The quality of the dataset is evaluated by measuring the model's performance (using mean absolute percentage error) on a separate validation dataset constructed from larger networks.

3.3.2 Proposed Data-centric Solution

The proposed procedure implements a customized three-step solution, shown in Figure 3.2. The restrictions on sample topology size means that the training and target domains will have significant differences, so this is partly a domain generalization problem. First, we generated an initial data set where we match the input variables for the simulator that do not scale with topology with those of the given validation set, pushing the domain of our generated set closer to the target. Then, we refactored the generated data set limiting the bounds of some input parameters to constrain the training domain so it does not drift too far from the target. And finally, we designed a cleaning framework to keep only high-quality data, discarding samples that hinder performance. These steps are described in detail in the following subsections.

Initial Data set Generation

We started by conducting a detailed analysis of the validation set to allow us to form hypotheses about how specific parameters of the validation sets were generated. For example, we wanted to hypothesize about the assignment of traffic and service type to source and destination node pairs, the average path length, the link capacity values and how these capacities are assigned to the links, the node buffer sizes and how they

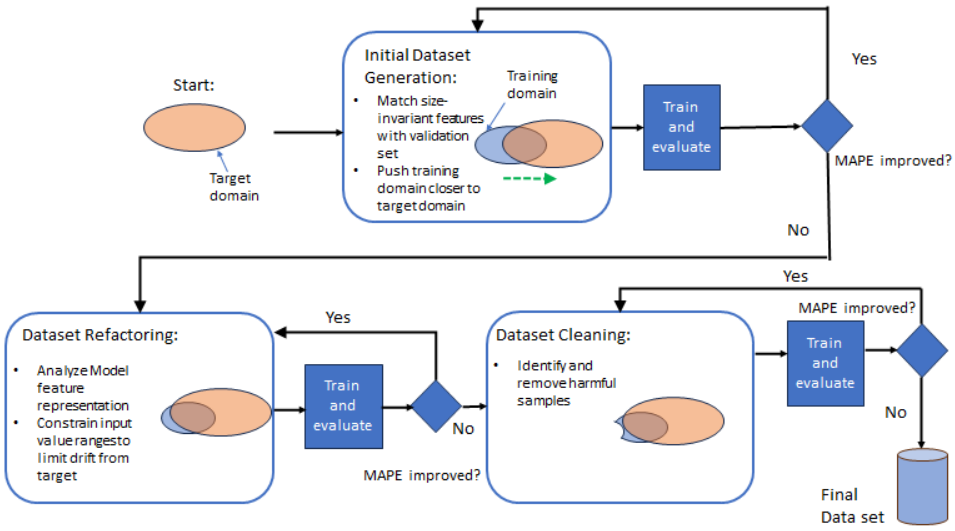


Figure 3.2: Proposed solution overview: a three-step process

are assigned, the scheduling policies, etc. Some of the parameters that we observed in the validation set are shown in Tables 3.1 and 3.2. In these tables, the ”values” column shows the set of possible values a parameter can take. The ”probabilities” column contains the probability of selecting a given value. For example, the parameter ”scheduling policy”, abbreviated to ”policies”, can be selected from the set of values [First In First Out (FIFO), Strict Priority (SP), Weighted Fair Queuing (WFQ), Deficit Round Robin (DRR)], each policy can be selected with equal probability, represented by [0.25, 0.25, 0.25, 0.25]. The other parameters in the tables can be interpreted in the same way, with values selected at random based on probabilities.

From the assumptions made on Tables 3.1 and 3.2, we can see that apart from the link capacity whose values change as the graph size increases, the other parameters seem to be chosen randomly with probabilities defined as in the tables. We made the assumption that we should use these values as input parameters for the simulator in order to generate our training set as well. So we generated a training set by choosing the different parameters as defined in Tables 3.1 and 3.2. After training the model with the data sets generated according to the process defined above, the best Mean Absolute Percentage Error (MAPE) we could obtain on the validation set was around 27-29%.

Table 3.1: Hypothesis made on graph nodes and edges after validation set analysis

Parameters	Values	Probabilities
Policies	[FIFO, SP, WFQ, DRR]	[0.25, 0.25, 0.25, 0.25]
Buffer Sizes	[8000, 16000, 32000, 64000]	[0.25, 0.25, 0.25, 0.25]
WFQ weights	[”70,20,10”,”33.3, 33.3, 33.4”,”60,30,10”,”80,10,10”,”65,25,10”]	[0.2, 0.2, 0.2, 0.2, 0.2]
DRR weights	[”60,30,10”,”70,25,5”,”33.3,33.3,33.4”,”50,40,10”,”90,5,5”]	[0.2, 0.2, 0.2, 0.2, 0.2]
Link capacity	[10000, 25000, 40000, 100000, 250000, 400000, 1000000]	based on number of nodes

Table 3.2: Hypothesis made on flows after validation set analysis

Parameters	Values	Probabilities
Traffic flow	Uniform(0,1)× I, I∈ [1000,2000,3000,4000]	[0.25, 0.25, 0.25, 0.25]
Packet Size Distribution	"0,500,0.22,750,0.05,1000,0.06,1250,0.62,1500,0.05"	0.2
	"0,500,0.08,750,0.16,1000,0.35,1250,0.21,1500,0.2"	0.2
	"0,500,0.53,750,0.16,1000,0.07,1250,0.1,1500,0.14"	0.2
	"0,500,0.1,750,0.16,1000,0.036,1250,0.24,1500,0.14"	0.2
	"0,500,0.05,750,0.28,1000,0.25,1250,0.27,1500,0.15"	0.2
Time Distribution	"Poisson","CBR","ON-OFF (5,5)"	[1/3, 1/3, 1/3]
Type of Service	0,1,2	[0.1, 0.3, 0.6]

Refactoring the Data set

We were unable to get satisfactory performance when trying to match the validation set's parameters with networks that had less than 10 nodes. We concluded that our data set was insufficiently diverse, i.e., that the parameters were not sufficiently variable to account for all potential use cases encountered during testing. We thus looked at other characteristics that could have an impact on the parameters defined in Subsection 3.3.2, such as delay ranges, queue utilisation, link utilisation, and average port occupancy.

Link Capacity Based on Routing:

Our exploration of the validation set showed that the average link bandwidth in a given topology generally increased proportionally with the number of nodes. We assume this is done to accommodate the increasing number of flows in larger topologies, as the validation follows the same rules as the training set, in that each source-destination pair of nodes has a single flow assigned to it. As we can only have one flow per node pair in topologies of our sample, and with the restriction that the topologies in the training set must not exceed 10 nodes, this implies that the total number of flows will always be much lower than the total found in the validation set. The link utilization analysis we performed on the generated data sets from Subsection 3.3.2 showed that our link utilization was very low on average (about 40%). In order to adjust the link utilization so that it covers a wide range, we set the link capacity based on the routing, as this gives us greater flexibility, as explained below. In this strategy, the routing policy is known in advance and we set the capacity of the links to be proportional to the traffic it encounters. We can therefore set the capacity so that we get a desired level of link utilization. Equation (3.1) shows how we obtained the link capacities.

$$\text{capacity}_\ell = \frac{\sum_{f \in N_\ell} t_f}{LU_\ell}, \tag{3.1}$$

where t_f is the traffic of the flow f, N_l is the number of flows traversing link l, and LU_ℓ is the link utilization

Algorithm 3.1 get_load()

mean $= [0.2, 0.4, 0.6, 0.8]$
weights $= [0.3, 0.3, 0.3, 0.1]$
while True **do**
    $\mu =$ random(mean, weights)
    $\sigma = \mu/2$
    $LU_\ell = \mathcal{N}(\mu, \sigma)$
    if $0 \leq LU_\ell \leq 1$ **then**
        break
    end if
end while
Return: LU_ℓ

of link l which was chosen from a normal distribution of mean μ and standard deviation $\sigma = \frac{\mu}{2}$. We have chosen the values of the mean μ such that the links experience low, medium and high utilization levels, similarly to what we have observed in the validation set. Algorithm 3.1 details the process for obtaining the link utilization LU_ℓ. Notice that the function "random(.,.)" chooses a value from vector "mean" with probability "weights". Choosing LU_ℓ this way adds variability to the data set, so each example yields a more meaningful contribution.

Using this approach, the link capacity values concentrated around 10000, which is minuscule compared to link capacities in the validation set. Looking at how the RouteNet_Fermi model [Ferriol-Galmés et al., 2023a] estimates the delay of a flow as follows,

$$\text{delay}_f = \sum_{(q,\ell)\in f} \left(\frac{\text{GNN\_}qo_\ell}{capacity_\ell} + \frac{\mu\_fps_f}{capacity_\ell} \right), \tag{3.2}$$

we see that the delay is highly dependent on the link capacity, as it is used to calculate the queuing and transmission delay. Since the GNN readout output $(\text{GNN\_}qo_\ell)$ may give similar values in the training and validation set, having a link capacity value in the training set that is very different from those in the validation set is counterproductive. To address this, instead of using the link capacity directly derived from Equation (3.1) (hereinafter referred to as 'cap'), we set the link capacity to be the closest candidate to 'cap' from the set {10000, 25000, 40000, 100000, 250000, 400000}, which is a subset of the link capacity in the validation set. Algorithm 3.2 details the process for setting the link capacities in the network according to the traffic information.

Network Topology Choice

To get the most flows per sample, we decided to only generate 10-node graphs, the maximum permitted. We found that having one unique network topology per sample, so 100 different network topologies in

Algorithm 3.2 set_link_bandwidth()

Input = G, paths, traffic, capacity_set
link_bw = Array of 0
for pair (SRC, DST) $\in G$ **do**
    path = paths(SRC,DST)
    for e in path **do**
        link_bw(e(SRC), e(DST)) += traffic(SRC,DST)
    end for
end for
for e in G.edges **do**
    $link\_load = get\_load()$
    cap = link_bw $(e(\text{SRC}), e(\text{DST}))$ / link_load
    $Id\_cap = \arg\min(|capacity\_set - cap|)$
    $G(e[\text{SRC}], e[\text{DST}]) = capacity\_set[Id\_cap]$
end for
Return: G

total, was not beneficial, as it could add too much noise to the data set (Achieved MAPE $\approx 16\%$ on the validation set with 100 different network topologies) . Therefore, we carefully designed and selected 10 network topologies to have two important characteristics found in the validation set: the presence of nodes of degree 1 and different node degree configurations, as shown in Figure 3.3 and Table 3.3, to increase the probability of having a different path length distribution using the shortest path algorithm. Indeed, on a 10-node graph, the presence of higher-degree nodes, e.g., 6, limits the path length using the shortest-path algorithm to around 3. Since we wanted to have different path length distributions using the shortest path algorithm, generating graphs on the basis of node degree gave us greater flexibility. Each topology was used to generate 10 samples where only the node and edge characteristics were modified. The node attributes were chosen as described in Table 3.1 and the edge characteristics were chosen according to Subsection 3.3.2.

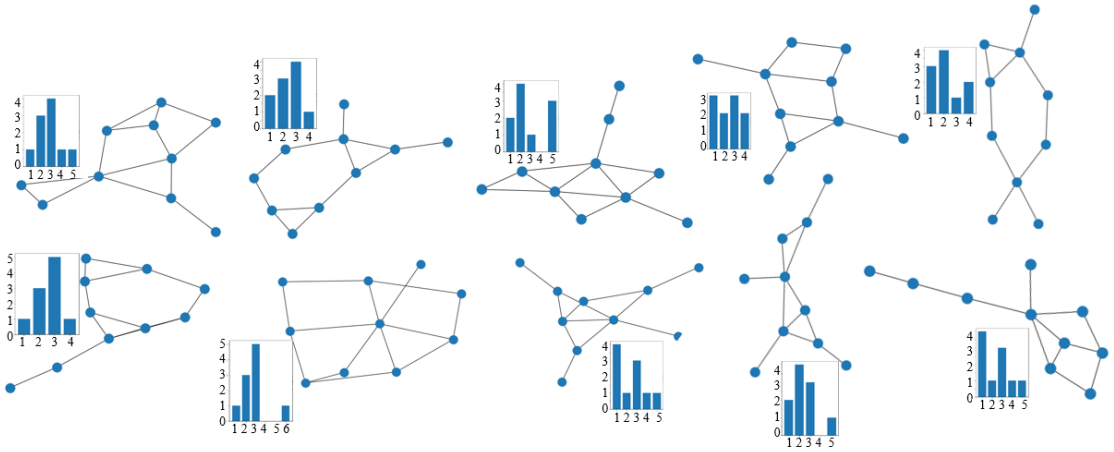


Figure 3.3: Selected network topologies with corresponding degree histogram (degree on x-axis, count on y-axis).

Flows Generation:

Table 3.3: Key topology characteristics.

Topology ID	1	2	3	4	5	6	7	8	9	10
$\|N\|$	10	10	10	10	10	10	10	10	10	10
$\|L\|$	14	12	14	12	11	13	14	12	12	12
Diameter	4	4	4	4	5	5	3	4	5	5
Avg. degree	2.8	2.4	2.8	2.4	2.2	2.6	2.8	2.4	2.4	2.4
Max. degree	5	4	5	4	4	4	6	5	5	5
Min. degree	1	1	1	1	1	1	1	1	1	1
Avg. betweenness	0.12	0.15	0.13	0.15	0.17	0.16	0.11	0.14	0.16	0.16
Max. betweenness	0.47	0.38	0.47	0.38	0.47	0.42	0.52	0.52	0.68	0.64
Min. betweenness	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Based on our observations of the validation set, we identified four traffic flow intensities: 1000, 2000, 3000, and 4000 bps, to use as a base. For each sample, we randomly chose an intensity and the flows in that sample were selected from the interval [intensity/2, intensity]. We also found that defining only 10 distinct traffic matrices delivers better performance than 100 different traffic matrices when working with 10 network topologies of 10 nodes each. Thus, we defined 10 different traffic matrices and used these same traffic matrices for each of the different network topologies. In addition, updating the probability distribution values of the ”packet size distribution” and ”time distribution” parameters we defined in Subsection 3.3.2 from [0. 2, 0.2, 0.2, 0.2, 0.2] to [0.1, 0.2, 0.3, 0.3, 0.1] and from [1/3, 1/3, 1/3] to [0.8, 0.1, 0.1] respectively resulted in flows that produced better results. These new values were found after several search cycles of our algorithm, as shown in Figure 3.2. We observe improvement when we generate more flows with a Poisson distribution (probability = 0.8) when we have few samples. In order to easily visualize the similarities between the training and the validation data set in terms of average flow value we plot Figure 3.4, in which we can see that after all the parameter tuning used to generate the training data set, both violin plots look alike, demonstrating the similarity between the training and validation sets in this particular aspect.

Training the model with the data set generated after the different changes described in Subsections 3.3.2, 3.3.2 and 3.3.2 resulted in a MAPE reduction from 27-28% to 8-10% on the validation set.

Data set Cleaning

With 100 samples generated, we explored whether cleaning this data set by removing some samples judged too noisy can help improve the results. As what was ”Noisy” was difficult to specify, we hypothesized on what can be considered noise.

Hypothesis 1: Noise based on data distribution

Our first hypothesis is: A sample is considered noisy if it contains certain characteristics (e.g., average

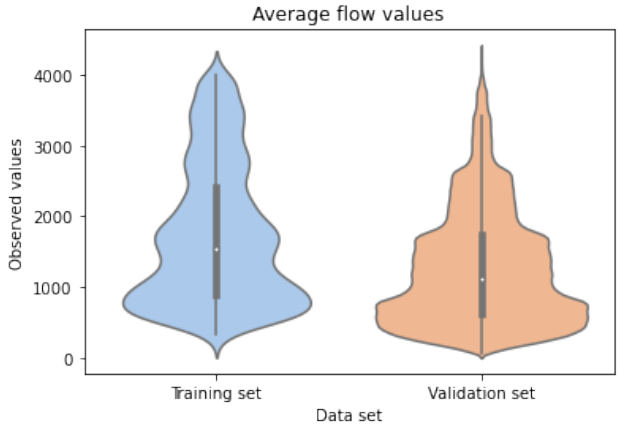


Figure 3.4: Traffic flow distribution: training vs. validation set.

port occupancy in our case) that are outside the validation set's distribution. We suppose this based on our knowledge of the RouteNet-Fermi model, which predicts the average port occupancy, then derives its delay prediction. Thus, we wanted the average port occupancy distribution of our generated training samples to match that of the validation set. When we cleaned our data set following this hypothesis, we saw no improvement, and could not confirm our first hypothesis.

Hypothesis 2: Noise based on path length

As we were using the shortest path routing algorithm and that message passing iterated 8 times, in order to reduce the over-smoothing during the training, we made a second hypothesis. We suppose that a sample is noisy if it does not contain at least one path of length 4, roughly the average path length in the validation set. We identified 10 samples and found that all these samples were generated with the topology containing a degree 6 node. After these samples were removed, we trained the neural network and the MAPE dropped to 6-7% over the entire validation set, using only 90 samples. We thus demonstrated how we use our understanding of the model to optimize our data set.

The pseudo codes describing how the network topologies, the traffic matrices and the final data set have been generated are given in Algorithm 3.3, Algorithm 3.4 and Algorithm 3.5, respectively.

3.3.3 Numerical Results

This section presents the results obtained by our methodology, compares the best solutions (in the challenge ranking) to our proposed solution, and highlights our findings. The experiments have been done on a machine running a Windows system equipped with an AMD processor of 32-core, 2.95 Ghz with 128

Algorithm 3.3 generate_topology()

Input = num_nodes, prob, policies, buffer_sizes,wfq_weights, drr_weights
$traffic$ = matrix of zeros of size G
for (SRC,DST) pair $\in G$ **do**
    node_schedulingPolicy = random(policies)
    node_bufferSizes = random(buffer_sizes)
    if node_schedulingPolicy == WFQ **then**
        wfqWeight = random(wfq_weights)
    end if
    if node_schedulingPolicy == DRR **then**
        drrWeights = random(drr_weights)
    end if
end for

Algorithm 3.4 generate_traffic()

Input =G, $intensity$, time_dists,td_weights, size_dists, sd_weights, tos_list, tos_weights
$traffic$ = matrix of zeros of size G
for (SRC,DST) pair $\in G$ **do**
    b_{avg} = random($[intensity/2, intensity]$)
    td = random(time dists, td weights)
    sd = random(size dists, sd weights)
    tos = random(tos list, tos weights)
    $traffic$[SRC, DST] = b_{avg}
end forReturn: $traffic$

GB of RAM and an NVIDIA GeForce RTX 2080 Ti. The code of our solution and the generated dataset is available on GitHub[1].

Results Analysis

The improvements obtained with our pipeline at each stage are shown in Table 3.4. We present the MAPE obtained on the entire validation set referred to as (”ALL”) as well as the MAPE obtained in each subset of the validation set grouped according to the number of nodes in their network topology (”50 nodes”, ”75 nodes”, etc.). As described in Section 3.3.2, we can see that we moved from a MAPE of about 28% to around 6% for the entire validation set. We identified some data sets (e.g., those with 170, 200 and 240

[1]https://github.com/ITU-AI-ML-in-5G-Challenge/ITU-ML5G-PS-002-SNOWYOWL-GNNetworking-Challenge2022

Algorithm 3.5 generate_traffic()

Input =num_nodes, prob, capacity_set, policies, buffer_sizes, wfq_weights, drr_weights, time_dists, td_weights, size_dists, sd_weights, tos_list, tos_weights, intensity_set
for $i \in$ {1,2,...,10} **do**
    for $j \in$ {1,2,...,10} **do**
        G = generate topology()
        $paths$ = shortest paths routing
        $intensity$ = random(intensity_set)
        $traff\_ar$ = generate_traffic()
        set_link_bandwidth()
    end for
end for
generate_dataset_simulator() ▷ use omnet++
clean_dataset() ▷ as described in Section 3.3.2

Table 3.4: MAPE (%) obtained on validation sets

Updates	50 nodes	75 nodes	100 nodes	130 nodes	170 nodes	200 nodes	240 nodes	260 nodes	280 nodes	300 nodes	ALL
section 5.1	27.89	26.66	29.31	32.83	34.43	34.06	31.78	17.32	28.29	25.23	28.26
section 5.2	7.66	8.19	9.01	10.93	10.94	12.34	10.91	3.85	7.45	6.97	8.57
section 5.3	6.00	6.22	6.81	7.97	8.47	8.48	8.02	3.16	6.10	5.40	6.50

nodes) as difficult, with higher MAPE at each stage. We note that the data set with the lowest MAPE at each stage (260 nodes) has a low number of samples (9) compared to 14 in other sets. As a baseline, the competition organizers stated that with thousands of samples of networks up to 10 nodes, they were able to get a MAPE of about 5% on the validation set. With our 90 generated samples, we reached a MAPE around 6%. This demonstrates that it is possible to efficiently train a neural network with smaller data sets if the data is of good quality.

Comparison Against Other Solutions

When comparing our solution to other finalists at the end of the challenge, a common characteristic seen in all top solutions was the detailed analysis of the validation set to extra insights and generate the initial data set. Table 3.5 presents the results obtained by the top 3 teams of the competition on the test set (our solution ranked first).

Team Ghost Ducks[2]**:** The second-ranked team generated around 270K samples and trained two oracle models with about 85K samples from the generated samples. They extracted a vector representation (embedding) for each sample in the validation and training samples. This vector representation contains the path state, link state and length of each flow path. They then clustered the validation set and assigned each training sample to a cluster. Finally, they took the top k samples from each cluster to arrive at a set of 100 samples to get their final traning set. They obtained a MAPE of 8.554 % on the test set.

Team Net: The third-ranked team proposed a beta distribution-based leave-one-out sample ranking strategy. They built their initial data set by generating different distributions from the beta distribution. They then assign each sample a score indicating the quality of the sample by examining the impact of removing that sample from the set. They then select the top 100 samples to use as their training set. They obtained a MAPE of 9.79 % on the test set.

The uniqueness of our solution lies in the fact that we did not generate thousands of samples before

[2]https://github.com/ITU-AI-ML-in-5G-Challenge/ITU-ML5G-PS-002-GhostDucks-GNNetworking-Challenge2022

reducing to a smaller set. Instead, we focused on understanding the data and the model to refactor and clean the 100 generated initial samples. We achieved our best result with only 90 samples, further lending credence to the fact that data quality is more important that volume.

Table 3.5: Top solutions comparison on the test set

	Snowyowl (our)	Ghost Ducks	Net
# samples	**90**	100	100
MAPE (%)	**8.55334**	8.55446	9.97016

By generating synthetic packet-level data through targeted simulation, we enable ML models to learn from clean, well-labeled, and customizable environments. However, while simulation offers control and flexibility, it may not fully capture the messiness of real-world dynamics. To bridge this gap and ensure robustness under practical conditions, we now shift our focus to flow-level data generation, where we repurpose real-world datasets to construct meaningful network traffic data.

3.4 Use case 2: Flow-based Dataset Refactoring from Urban Data

In this section, we describe the proposed flow-based 5G-type predictable traffic generator that relies on the refactoring of open data of vehicle and pedestrian traffic from the City of Montreal [Dataset, 2022]. Indeed, the latter data is refactored in order to generate different classes of network traffic, with different characteristics associated with typical 5G applications, and then with different traffic patterns and peak hours. The result is a valuable traffic generation tool for researchers interested in validating machine learning algorithms aimed at, for example, traffic forecasting, resource elasticity, or automated scaling of slice resources.

3.4.1 5G Networking and Orchestration Environment

We now describe the key elements of the 5G network slicing environment for the development of a traffic generator.

5G E2E Network Slicing Environment

We consider a reference 5G network environment characterized by three segments: Radio Access Network, Transport Network and Core Network, see Figure 3.5. We add to this the network slicing component,

i.e. the ability to simultaneously deploy and use different dedicated virtual networks, each specialized in the provision of a given set of services and/or a set with given subscribers. In this context, SDN and NFV technologies play a critical role in the design of 5G network slices. Slices are virtual networks composed first of a 5G End to End (E2E) user plane flow and then, in the Data Network (DN) of one or more ordered Virtual Network Functions (VNF)s, defining a so-called Service Function Chain (SFC). Example of a SFC, e.g., for Voice over Internet Protocol (VoIP), is NAT $\rightarrow$ FW $\rightarrow$ TM $\rightarrow$ FW $\rightarrow$ NAT, with the following VNFs: Network Address Translation (NAT), Firewall (FW), and Traffic Monitor (TM), see [Askari et al., 2018].

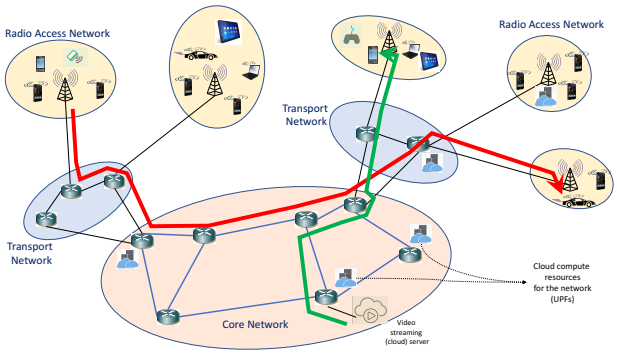


Figure 3.5: Reference 5G E2E network slicing environment

The 3GPP 5G RAN architecture [3GPP, 2018c] consists of a set of radio base stations (known as gNBs) connected to the 5G core network and to each other. The gNB includes three main functional modules: a RU, a DU, responsible for real time scheduling functions, and a CU responsible for non-real time ones. In a 5G cloud RAN, the DU's server and relevant software could be hosted on a site itself or can be hosted in an edge cloud (datacenter or central office). The CU's server and relevant software can be co-located with the DU or hosted in a regional cloud data center. In 5G RAN, at the network level, we distinguish three parts: Fronthaul, the link connectivity between the RU and DU ; Midhaul, the link connectivity between the DU and CU ; and lastly the Backhaul, the link connectivity between the CU and the core network. In the core network, the UPF performs user processing and transfers data. While control plane functions can be shared between network slices, UPFs are slice specific in terms of QoS requirements. Although a 5G network is often built as an assembly of different components and specialized networks, an E2E vision makes it possible to better understand the requirements of the 5G network and to provide more efficient 5G solutions to satisfy users' quality of service requirements and to meet operators' business needs.

In the sequel, we model the logical 5G network as a directed graph $G = (V, L)$, where V is the set of nodes and L is the set of logical links. A subset of the nodes is equipped with computer resources (e.g., servers for applications such as gaming/video streaming or datacenters for hosting UPFs), and are commonly called compute nodes. These nodes have processing capabilities in terms of VMs or containers where each VM/container is characterized by its number of CPUs (and their vCPU counts), RAM and storage.

5G Provisioning

A 5G request is characterized by a source and a destination in the logical network, start-up and hold time, bandwidth requirement, end-to-end delay, and the required SFC. The source or destination can be the location of a User Equipment (UE) or the source (server location) of, e.g., a downstream video stream. Latency of a request has two components: *(i)* the network component with its four parts, i.e., propagation, transmission, queuing delay, and processing delays, both in the Core Network (CN) and in the Radio Access Network (RAN). *(ii)* the software component with the processing times associated with the various 5G logical funct bidirectional)

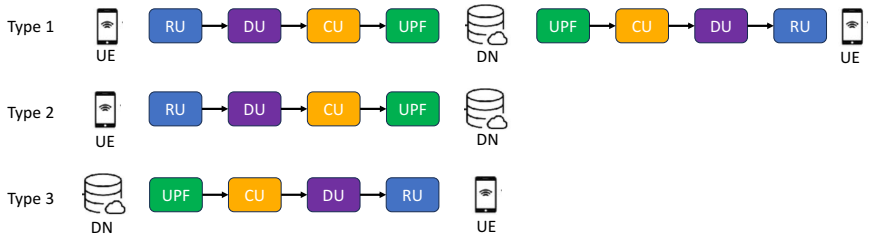


Figure 3.6: Three types of 5G user plane flows

as described in Figure 3.6. We will not go through the details of the functions embedded in each logical functionality, with the understanding that these functions perform according to the QoS requirements. Each entity, RU, DU, CU and UPF has its own hardware/software requirements. Software requirements are translated into compute requirements, which vary with the class of traffic, in terms of CPU, Random Access Memory (RAM) and storage resources. The latter ones are provided by the Virtual Machine (VM)s or the virtualized containers hosting the 5G logical functionalities.

3.4.2 Proposed 5G Traffic Generator

Over time, for example over a day, 5G traffic patterns change, both in terms of overall distribution and intensity, but also in terms of the nature of applications. While we see many work today with traffic prediction and elastic resource provisioning, there is a lack of data sets to test and validate the proposed algorithms. In this work, we attempt to provide a versatile traffic generator that can fill that gap.

Open Data Sets of the City of Montreal

Among the open data of the City of Montreal [Dataset, 2022], we used the traffic data with the counts of different types of vehicles, bicycles and pedestrians at a given number of street intersections. Measurements are taken at 15 minute intervals during certain times of the day and data is available from 2008 to present. The observations detail the start time of each count period, the number, origin and direction of vehicles, pedestrians and cyclists for each possible movement at an intersection and the geographical coordinates of the intersection. The counted entities are classified into sixteen categories: Cars, Light Trucks, Heavy Trucks, Pedestrians, Bicycles, Buses, Schoolchildren, Trucks, Straight trucks, Articulated trucks, Motorcycles, Unused, Uturn, Illegal, Other and All.

Different Traffic Patterns

To generate non-uniform and meaningful traffic distribution, we adapted the population gravity model used in [Betker et al., 2003] for various transport network traffic scenarios, with a refactoring (see Section 3.4.2 for the details) of the open vehicular and pedestrian data of the city of Montreal [Dataset, 2022].

The number of service requests per node pair is proportional to the product of the respective populations divided by the distance between them. Indeed, at time t, each network node $v \in V$ has a population (aka users) denoted by $N_v(t)$ and the traffic of node pairs $(v, v') \in V \times V$ is computed as follows:

$$P_{vv'}(t) = \frac{\log(100 + \frac{N_v(t)N_{v'}(t)}{D_{vv'}})}{\sum_{w \in V} \sum_{w' \in V} \log(100 + \frac{N_w(t)N_{w'}(t)}{D_{ww'}})}, \tag{3.3}$$

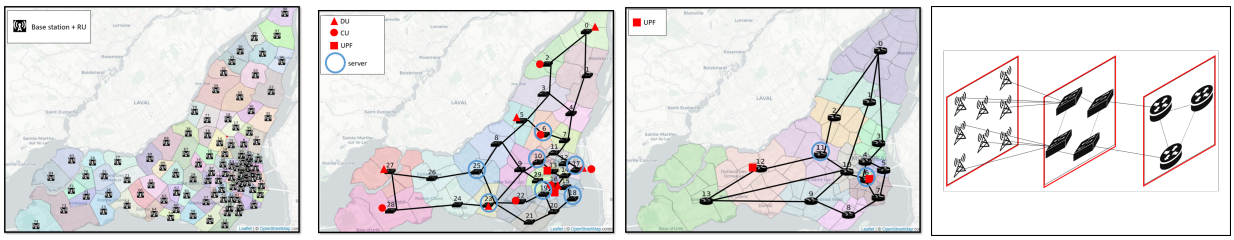
where $D_{vv'}$ is the geographical distance between nodes v and v', $v \neq v'$.

In order to allow requests within the same RAN, i.e., requests with the same source/destination node in the core network, we define D_{vv} as a scaling factor rather than a distance, and decompose $N_v(t) =$

$N_v^{\text{IN}}(t) + N_v^{\text{OUT}}(t)$ for the number of users with service requests within the same RAN, and between two different RANs, respectively. As a consequence, $P_{vv'}(t)$ is computed with $N_v^{\text{OUT}}(t)$ for $v \neq v'$, and with $N_v^{\text{IN}}(t)$ when $v = v'$. Note that the addition of the log factor is motivated by the need to smooth the significant unbalance among node data.

Network Environment

We design the logical layer of a E2E network relying on open data from the urban traffic data of the City of Montreal [Dataset, 2022]. It contains the traffic counting information of different categories of vehicles every 15 minutes at different traffic light intersections of the city of Montreal. Based on the geographical locations of the intersections, we clustered them into 100 cells, representing the radio base stations, see Figure 3.7(a). The base stations are connected to the transport network shown in Figure 3.7(b), which in turn is connected to the core network represented in Figure 3.7(c). Each base station node network is abstractly associated with a set of UEs, each of which is associated with one or several types of applications and its typical access delay. RUs are hosted at the base station, while DUs and CUs reside in the edge cloud of the transport network.. To accommodate latency-sensitive applications (e.g., URLLC slices), some UPFs are hosted in the edge cloud while others are located in the central cloud. The resulting network is shown in Figure 3.7, where some servers running applications such as gaming/video streaming are depicted. Figure 3.7(d) illustrates how the different network components are connected.



(a) Radio network (b) Transport network (c) Core network (d) Network hierarchy

Figure 3.7: 5G network: radio, transport and core infrastructures and their hierarchy

Sequence of Logical Nodes and VNFs

We considered 6 types of services and their corresponding sequence of logical nodes. Behind every logical nodes, there is a sequence of one or several VNFs that we do not detail. We reuse some of the latency and bandwidth requirements reported in [Askari et al., 2018]. We adapted some values, in particular

Table 3.6: 5G user plane flows (bandwidth and latency values adapted from [Askari et al., 2018])

Services	E2E user plane flows	Bandwidth per user or IoT system	E2E latency	Request bundles
Cloud gaming	UPF_{CG} - CU - DU - RU	4 Mbps	80 ms	[40-55]
Augmented reality	UPF_{AR} - CU - DU - RU	100 Mbps	10 ms	[1-4]
VoIP	RU - DU - CU - UPF_{VOIP} - UPF_{VOIP}- CU - DU - RU	64 Kbps	100 ms	[100-200]
Video streaming	UPF_{VS} - CU - DU - RU	4 Mbps	100 ms	[50-100]
Massive IoT	RU - DU - CU - UPF_{MIOT}	[1-50] Mbps	5 ms	[10-15]
Industry 4.0	RU - DU - CU - $UPF_{I4.0}$ - $UPF_{I4.0}$ - CU - DU - RU	70 Mbps	8 ms	[1-4]

Table 3.7: CPU/RAM/Storage core usage for 5G logical functionalities

5G logical functionalities	vCPU	RAM (Gb) per 100 Mbps	Storage (Gb) per 100 Mbps	5G Logical functionalities processing time per 0.01 msec unit
RU	1	4	7	12
DU	9	5	1	6
CU	11	15	2	22
UPF_{CG}	13	15	7	14
UPF_{AR}	5	2	5	16
UPF_{VOIP}	5	2	5	2
UPF_{CG}	5	11	10	4
UPF_{MIOT}	5	3	20	4
$UPF_{I4.0}$	5	4	11	4

those of Massive Internet of Things (MIoT), using, e.g., [Mtawa et al., 2019]. We provide the typical E2E requirements, with augmented reality and Industry 4.0 (smart factory) sharing the most stringent ones, see, e.g., [Walia et al., 2019]. Table 3.7 provides the compute resource modeling that we use with a required amount of CPU (in terms of percentage of CPU per user), RAM and Storage for each logical node. These values do not come from real use cases (due to lack of access to real data) and only provide a certain order of magnitude. We considered 6 different slices as illustrated in Table 3.6 with different bandwidth and latency requirements.

Network Dimensioning

Another important aspect is the dimensioning of the network links, as it impacts the provisioning of the service requests, their access to compute resources and then the end-to-end delay. Having in mind that most of the network operators have enough capacity to grant most of the requests if not all on heavy traffic periods (peak times) and that the transport capacities are set to last for a particular duration (e.g., a few weeks to a few months), we dimensioned the links and compute nodes capacities in such a way that GoS

is acceptable even in heavy traffic periods. Indeed, we use data from the busiest traffic periods of the city of Montreal to dimension the network. Starting with a network with unbounded link capacity and compute node resources, we routed all the busiest period generated data on the shortest path and recorded for each link and compute node their maximum resource usage. The shortest path was run on the multi-layer graph [Huin et al., 2018] that is commonly used for the provisioning of 5G service requests with SFCs, with links weighted by network delays and cross-layer links weighted by the 5G logical functionalities processing times. We then set each transport capacity value to a number uniformly selected from the range [90, 110]% of its corresponding maximum usage within the shortest path usage. This is to enforce that the shortest path will not always be used.

Regarding the E2E delay, we defined it carefully so that we get a reasonable delay for each application and request. Tables 3.6 and 3.7 provide the resulting values.

Request Generation for a Given Distribution

As explained in Section 3.4.2, we refactor the data of the City of Montreal in order to use them to simulate different 5G slices, each with a different traffic pattern and intensity over the days. The resulting slice traffic, although not necessarily representative of the associated application, corresponds to real data, i.e., a good fit for validating and testing machine learning algorithms for, e.g., traffic prediction or elastic resource management.

Considering the urban traffic of the city of Montreal, we constructed 6 artificial slices (sequences of logical nodes in the context of a 5G network) by combining some urban traffics as presented in the Table 3.8. For each slice, we used the overall count of the associated vehicles/pedestrian at the street intersections at specific time stamps. Since the statistics are collected every 15 minutes in [Dataset, 2022], we end up with a dynamic vehicle/pedestrian model which, in turn, gives a dynamic and non-uniform traffic distribution, which is artificial as far as 5G applications are concerned, but still with real data behavior. The pattern of

Table 3.8: Mapping of urban traffic to 5G slices

Service chains	Slices	Vehicle/Pedestrian types
Video streaming	Slice 0	Cars
Cloud Gaming	Slice 1	Pedestrians + Schoolchildren
VoIP	Slice 2	all Trucks categories
MIoT	Slice 3	Bikes + Motorcycles
Industry 4.0	Slice 4	Buses
Augmented reality	Slice 5	all other categories

the original urban traffic data being different from the one of network traffic data like represented in, e.g., [Tarng et al., 2008] (gaming), [Rahman et al., 2018] (video streaming) for different applications, we used some scaling, shifting and transformation functions to change the urban traffic data and obtain a similar pattern with the network data. This way, we got organized so that video streaming (Slice 0), which is today the dominant traffic, has a large share of the traffic with peak hours late in the evening. Indeed, studies show that video traffic account for about 70 % in 2022, a share that is forecast to increase to about 80 % in 2027.

Using the data mapping described in Table 3.8, we first get a number of requests per slice using the number of items, i.e., vehicles or pedestrians. We then derive an estimate of the required bandwidth for each slice at each time period (every 15 minute) with the product of the number of user requests and the corresponding bandwidth BW^{SLICE} requirement associated with each slice. Using the gravity model that was described in Section 3.4.2, we next non uniformly distribute the traffic requests among the different node pairs, using the user scaling given in the last column of Table 3.6, i.e., each request is now a bundle of user requests.

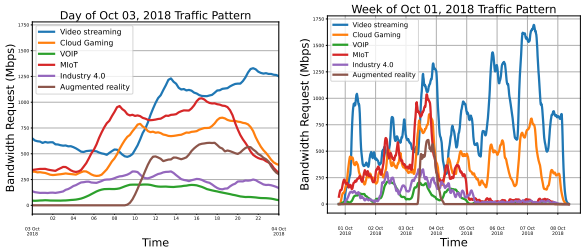
Another vital aspect of our dynamic traffic generator is to set the start t_i and end t'_i times of a request i (holding time $h_i = t'_i - t_i$). We generate or terminate a request based on the gravity model. Using the transformed data from the city of Montreal, we know for each time period the number of requests to generate per slice. The start time t_i of request i is then given by the transformed data of the city of Montreal. The holding time of a request h_i was selected using a geometric distribution with a mean of 5, from which the end time was then inferred. Each generated request is an aggregate of user-requests with the same application, the corresponding number of users per request was randomly selected in the range defined in the last column of Table 3.6.

3.4.3 Characteristics and Analysis of the Generated Traffic Data

We present here some characteristics of the generated traffic data.

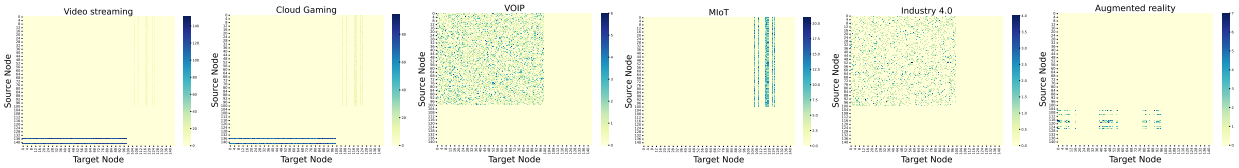
Different Weekly Patterns Depending on the Applications

An advantage of using our generator is that, as it is based on open data from the city of Montreal which is updated every 15 minutes, we can generate meaningful data from any selected time period with a very diverse load. Figure 3.8 presents the generated traffic pattern of each application of the week going from



(a) One daily traffic pattern (b) One weekly traffic pattern

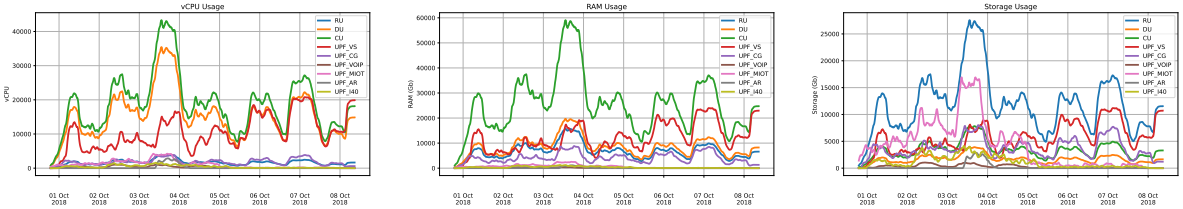
Figure 3.8: Traffic patterns



(a) Video Streaming (b) Gaming (c) VoIP (d) MIoT (e) Industry 4.0 (f) Augmented Reality

Figure 3.9: Heat maps of the bandwidth requirements of the requests

October 1st to October 7th, 2018. We observe that the traffic varies globally according to the days and the applications, video streaming and cloud gaming being the dominant applications with more than 50% of the traffic on the one hand and the least bandwidth-intensive VoIP on the other hand. Video streaming, cloud gaming and augmented reality have their peak at similar times, while different behaviours are observed for VoIP, Industry 4.0 and MIoT. Observed fluctuations are related to a real event (urban mobility), and knowing that in 5G, user mobility is an important aspect, the generated data can therefore help to train a machine learning model, when we expect a change in pattern and distribution.



(a) CPU (b) RAM (c) Storage

Figure 3.10: Compute resources over one week per logical node

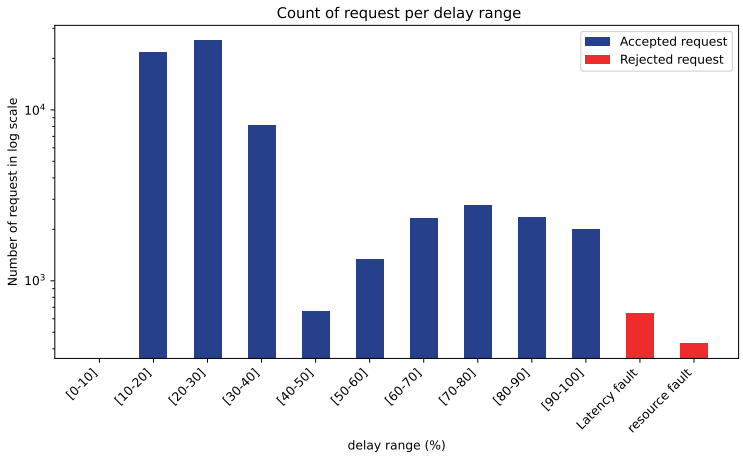


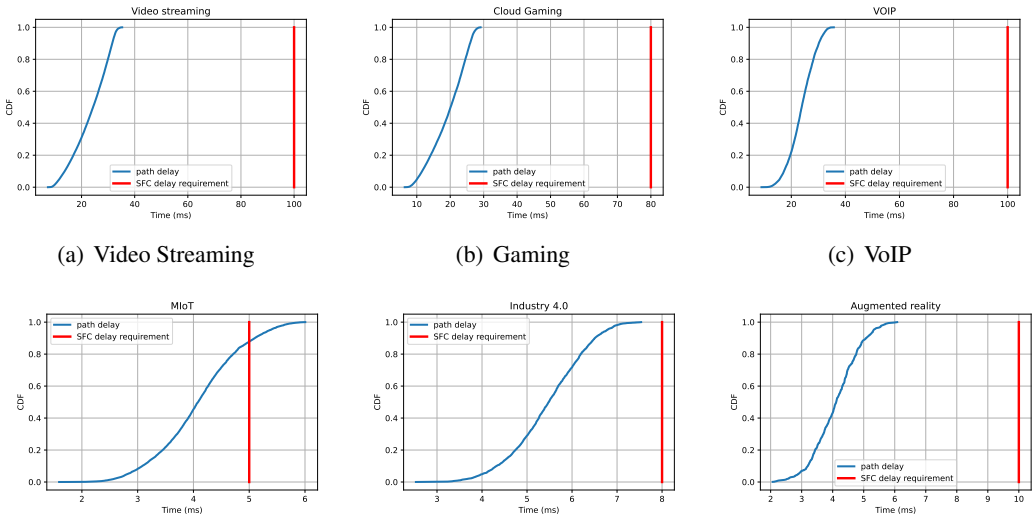
Figure 3.11: Count of requests per delay range, accepted vs. rejected.

Heat Maps

To visualize the distribution of the requests, Figure 3.9 illustrates the number of requests per node pair within the network. The illustration has been done per application, as they do not necessarily share the same source and target set. We can easily identify the three types of applications, server-to-base station downlink based applications where the source nodes are servers and the target ones are any base station node; base station-to-server uplink based applications where the source nodes are base stations and the target ones are servers; and finally anyone-to-anyone uplink and downlink based applications where the source/destination nodes can be any base station. Overall, we can see that the distribution of traffic is not uniform and is spread over the network nodes. All applications have a non-uniform distribution of demand based on their respective sets of sources and targets. Augmented reality applications have been located in the heart of the city of Montreal in our settings. Therefore, their corresponding distributions are only distributed on nodes located in the downtown area.

Compute Resources

To set a proper initial dimensioning of the compute resources based on the generated data, we used the shortest available weighted path and measured the resource utilisation for each service request. Figure 3.10 presents the amount of compute resources per logical node over the selected period. As expected, we can see a correlation between resource consumption and traffic load. Indeed, more resources were used on Oct 3^{rd} as it was the day with the most traffic in the data of the city of Montreal.



(a) Video Streaming (b) Gaming (c) VoIP

(d) MIoT (e) Industry 4.0 (f) Augmented Reality

Figure 3.12: CDF - Path delay vs delay requirements

The observed utilization patterns are highly dependent on the defined parameters. In spite of being able to find practical values for the various resource consumption, we define them arbitrarily, with linear consumption with respect to the bandwidth, while some of them maybe nonlinear [Rossem et al., 2019]. However, we set the values differently for the different UPFs to reflect the diversity of these values depending on the type of services. This usage is presented here to show that we can train models with our datasets to get some proof of concept.

Furthermore, as shown in Figure 3.11, some request flows are denied due to either latency faults or resource faults. This highlights the importance of proper network dimensioning and demonstrates that a greedy shortest path algorithm is insufficient to handle all request flows, suggesting the need for more sophisticated algorithms.

Path Delay vs. Delay Requirement

Delay requirements by application are illustrated in Figure 3.12 where we plot the Cumulative Distribution Function (CDF) of path delay versus application delay requirement. The processing delay of the various logical nodes is made proportional to the bandwidth requirements of the service requests. We observe that for applications with longer E2E, the overall delay is much smaller in % than for the applications with more stringent delays such as MIoT, Industry 4.0 or Augmented Reality. Applications with stringent delays are the ones most likely to encounter delay issues, with, e.g., MIoT suffering from rejection of requests due to

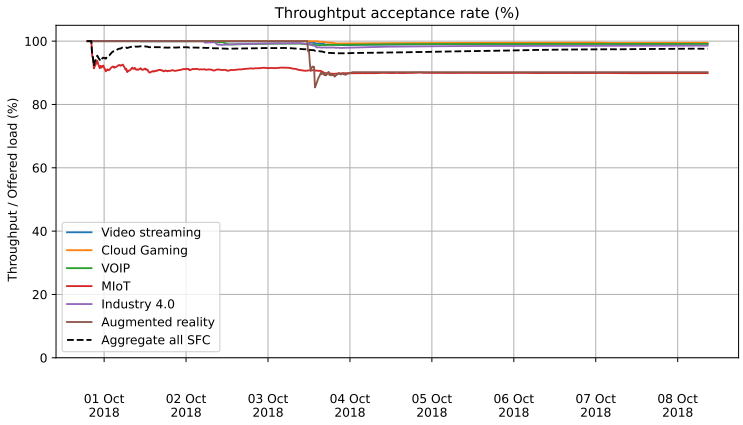


Figure 3.13: Grade of service

unmet delay requirements.

Grade of Service (GoS)

In Figure 3.13 we present the grade of service computed as the ratio of the throughput over the offered load. We observe that MIoT starts to decrease on Oct. 1^{st} and experiences the highest denial rate with a Grade of Service (GoS) around 84%. Augmented reality starts decreasing towards the middle of Oct. 3^{rd}, which represents the peak traffic time that was considered to dimension the network. The overall aggregated GoS, see the dashed line, is above 90%, indicating the quality of the proposal.

3.5 Conclusion

This work introduces a dual-strategy for network traffic data generation to overcome the scarcity of accessible, high-quality datasets in AI-driven networking. By combining simulation-based packet-level generation with real-world flow-level refactoring of mobility traces, we provide a flexible framework for diverse ML tasks. Packet-level data enables precise control and customization, while flow-level data captures real-world variability for robust model evaluation. Together, these approaches form a comprehensive foundation for developing and validating AI models under both controlled and realistic conditions.

Chapter 4

Energy Efficient Placement of Logical Functionalities in 5G/B5G Networks

This chapter addresses the joint optimization problem of routing and logical functionalities placement in 5G and beyond-5G (B5G) networks. Specifically, it formulates the deployment of Radio Units (RUs), Distributed Units (DUs), Centralized Units (CUs), and User Plane Functions (UPFs) as a large-scale energy minimization problem, subject to dynamic traffic distribution and end-to-end latency constraints. To tackle the complexity of the problem, a column generation-based decomposition approach is developed, complemented by an efficient practical heuristic to ensure tractability and scalability. The resulting solution provides a cost- and energy-efficient deployment strategy, forming the physical and operational foundation for subsequent intelligent control and orchestration.

4.1 Introduction

As 5G continues to evolve and we start planning for 6G, difficult questions remain regarding logical functionalities, considering cost, power, and latency requirements. On the one hand, it is good to host logical functionalities as close as possible to the radio access network (RAN), to meet latency requirements, e.g., the latency requirements of ultra-reliable low-latency communications (uRLCC) (from less than 1 millisecond (ms) to about 50 ms) or the increase in global 5G traffic (mobile data traffic has increased nearly 300-fold in the last 10 years). One the other hand, we have to deal with the enormous energy consumption of data centers involved in the different (edge) clouds hosting the growing software part of 5G networks. It is then

easier to save energy by moving away from the RAN to share computing resources. The question then arises: how many computing resources should be provided close to the RAN to support logical functionalities and their strict delay requirements, and how far away should they be provided to be able to increase sharing and meet user needs at any time, taking into account traffic fluctuations.

The placement of logical functionalities has been studied in different contexts in the literature. Most of the publications on placement deal with the placement of Virtual Network Functions (VNFs) in data networks, in which traffic flows need to go through a given sequence (called service chain) of strictly or partially ordered VNFs, which vary from one service to service, e.g., [Huin et al., 2018; Sun et al., 2022]. An increasing number of publications now deals with the placement of 5G logical functionalities (DU, CU and, but not always, UPF), with the same service chain DU-CU-UPF for all services, except for their delay requirements, e.g., [Ziazet et al., 2023; Klinkowski, 2020, 2024]. Finally, a new set of recent publications deals with the placement of microservices, which, like VNF placement, must obey a strict or partial order for traffic provisioning, see, for example, [Ding et al., 2023]. Despite the three different network contexts, these placement problems are very similar from a mathematical modeling perspective, and the literature still lacks an integrated optimization framework that holistically considers all these virtual functions in a multi-site multi slice architecture.

We study here the energy-aware placement of logical functionalities, DU, CU and UPF from an E2E network point of view, borrowing and adapting some of the mathematical tools proposed for VNF placement, in particular those with large-scale optimization techniques [Huin et al., 2018]. We propose a comprehensive mathematical model which integrates all the key constraints and solve it using large-scale optimization techniques, i.e., with mathematical decomposition algorithms. The decomposition algorithm is complemented by an efficient practical heuristic to ensure tractability and scalability.

4.2 Related Work

While extensive research has addressed Virtual Network Function (VNF) placement and Service Function Chain Placement (SFCP) in traditional data networks [Huin et al., 2018; Hyodo et al., 2019], comparatively less attention has been given to the placement of logical network functionalities specific to 5G and beyond (B5G) systems. These include DUs, CUs, and UPFs, which play key roles in the disaggregated 5G architecture. Although some studies focus on individual functions such as DU/CU [Liu et al.,

2020; Klinkowski, 2020; Askari et al., 2020] or UPF [Leyva-Pupo et al., 2019], a comprehensive treatment that jointly considers all logical functionalities under practical constraints remains largely unexplored. This section surveys relevant literature from the broader VNF/SFC domain and progressively narrows to works directly addressing DU, CU, and UPF placement in 5G/B5G environments.

4.2.1 VNF and SFC Placement in Traditional Networks

The VNF and SFCP problems have traditionally been tackled using three main approaches: mathematical optimization, heuristic algorithms, and machine learning techniques. Each has its strengths and limitations, particularly with respect to scalability and real-world applicability.

Mathematical Programming Approaches

Many studies formulate SFCP as an optimization problem using models such as Integer Linear Programming (ILP), Mixed Integer Linear Programming (MILP), or Binary Integer Programming (BIP) [Li et al., 2019; Qi et al., 2019; Tang et al., 2018; Ghazizadeh et al., 2022; Pei et al., 2018, 2019]. In [Kim et al., 2016], authors proposed an energy-efficient SFC placement algorithm that ensures Quality of Service (QoS) while while minimizing energy consumption. Tha algorithm first compute latency-constrained paths and then VNFs are deployed accordingly. It also offers reconfiguration of SFC when energy consumption exceeds a given threshold value. However, the limited types of VNFs restrict its practical applicability. [Huin et al., 2018] introduced a scalable exact model using decomposition to reduce latency and bandwidth usage, solving instances of up to 50 nodes efficiently. The proposed framework avoids data passing through unnecessary devices in order to minimize the bandwidth consumption and end-to-end latency. [Pei et al., 2018] modeled dynamic VNF deployment using BIP, optimizing latency and resource utilization.

These methods offer optimal solutions in small networks but are computationally intensive and less effective at scale.

Heuristic Methods

Heuristic techniques are widely adopted to handle the complexity of large-scale SFCP problems. In [Bari et al., 2016], authors used dynamic programming to orchestrate VNFs and demonstrated reduced operational costs in real network scenarios. Similarly, [Mechtri et al., 2016] proposed a greedy heuristic

based on feature decomposition, effectively addressing the connection constraints of SFCs.

Though efficient, heuristic methods often lack theoretical guarantees of global optimality and may yield suboptimal placements.

AI and Machine Learning-Based Approaches

Machine learning has emerged as a promising tool for addressing SFCP, offering adaptability and data-driven optimization. In [Pei et al., 2019], authors employed Double Deep Q-Networks (DDQN) to learn optimal VNF placements in SDN/NFV-enabled networks. The evaluation showed that the algorithm has excellent performance in throughput, delay and load balancing. However, this approach utilizing feedforward neural networks in the Deep Reinforcement Learning (DRL) algorithm could not accurately acquire network topology information and exhibited limitations in generalization when confronted with network topology changes. [Sun et al., 2020] improved generalization by combining DRL with Graph Neural Networks (GNNs) in the DeepOpt algorithm. However, evaluations were conducted on relatively small networks (23 nodes), limiting their relevance to larger-scale deployments. [Wang et al., 2020] addressed latency-aware SFCP in Mobile Edge Computing (MEC) using a DRL-based job scheduling approach. The aim was to minimize system latency, by demonstrating the effectiveness of service function chain (SFC) scheduling. Similarly, [Liu et al., 2022] proposed a reinforcement learning-based multi-objective optimizer for SFCP, balancing resource consumption, revenue, acceptance rate, and latency. They modeled the SFCP as a Markov decision process (MDP) and used a two-layer policy network as an intelligent agent. Simulation results showed that the proposed algorithm shows excellent performance in terms of underlying resource allocation revenue and VNF acceptance rate.

Although promising, AI-based approaches require further testing in realistic, large-scale 5G/B5G scenarios to prove their viability.

4.2.2 Placement of Distributed and Centralized Units (DUs/CUs)

With the disaggregation of the Radio Access Network (RAN) in 5G, the placement of DUs and CUs significantly impacts latency and energy consumption. [Klinkowski, 2020] introduced a latency-aware MILP model for DU/CU placement within fronthaul networks. Their model ensures that latency constraints are satisfied under various deployment scenarios. hey conducted a series of experiments across diverse network

scenarios to evaluate the efficacy of the MILP approach and analyze its impact on network performance. In [Xiao et al., 2021], authors jointly optimized DU/CU placement and optical path selection using MILP and proposed a heuristic for complexity reduction. Energy savings were achieved by consolidating processing functions and optimizing transport paths. Authors in [Zorello et al., 2022] tackled energy-aware DU/CU placement using both MILP and heuristic algorithms. Their models respected capacity and latency constraints and outperformed fully decentralized RAN deployments in energy savings. [Askari et al., 2020] proposed a dynamic DU/CU assignment heuristic based on traffic patterns, achieving energy savings with negligible service impact. Their objective was to centralize baseband functions in selected nodes, based on traffic flows, with the aim of minimizing node energy consumption. This consolidation had to meet constraints, particularly in terms of link capacity and front-end latency. The work presented in [Li et al., 2022b] extended the problem to include MEC node placement, formulating a MILP for joint DU, CU, and MEC deployment. A heuristic was also provided to ensure scalability.

4.2.3 Placement of User Plane Functions (UPFs)

UPFs are critical for 5G core networks and MEC, yet their placement remains underexplored. The work in [Leyva-Pupo et al., 2019] developed two MILP models for optimal UPF placement and relocation minimization, achieving improved reliability and latency compliance through backup sharing and mobility-aware design. Specifically, a substantial reduction in the number of UPFs required can be achieved by sharing the capacity of backup UPFs, and UPF displacement can be minimized by taking user mobility into account and distinguishing between main UPFs and their backup counterparts. In follow-up work, [Leyva-Pupo and Cervelló-Pastor, 2022] addressed UPF chaining in MEC environments using a multi-objective ILP model. They proposed the placement and chaining UPF (PC-UPF) heuristic and a simulated annealing-based metaheuristic to reduce provisioning costs while satisfying QoS constraints. Both approaches demonstrated near-optimal performance with faster runtimes.

Summary and Gaps

The existing literature shows significant progress in addressing VNF/SFC placement in conventional networks and some aspects of DU, CU, and UPF placement in 5G. However, most studies consider only a subset of functionalities or focus on isolated objectives (e.g., energy, latency). There is a clear research gap

in jointly addressing the placement of all major logical components (RU, DU, CU, UPF) within a unified framework that considers real-world 5G constraints, scalability, and traffic dynamics. We believe that an E2E placement approach offers substantial benefits by providing a holistic perspective of the network.

4.3 DU/CU/UPF Placement and 5G Provisioning

We describe here a concise definition of the problem statement with respect to DU/CU/UPF placement and 5G provisioning, as well as an adaptation of the so-called layer graph for the calculation of lightpaths satisfying the routing constraints through the DU/CU/UPF logical functionalities. Additionally, we introduce an energy model that captures the consumption behavior of the two key components influencing the placement decisions.

4.3.1 Notation and Network Architecture

We consider a 5G Software-Defined Network (SDN) that is represented by a graph $G = (V, L)$ where V represents the set of nodes and L the set of physical links. We assume that only a subset of nodes $V^f \subseteq V, f \in \{\text{DU}, \text{CU}, \text{UPF}\}$ are connected to a processing pool (cloud) that can process some 5G logical functionalities, i.e., DU, CU or UPF [3GPP, 2021]. Following ITU-R, 5G traffic covers three traffic categories or classes of services (CoS): Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communications (URLLC), and Massive Machine Type Communications (mMTC). On the user plane, each traffic flow request follow the same logical functionality chain as described in Figure 4.1: only the direction varies according to uplink or downlink requests. Indeed, the only difference for the logical functionality chain is related to the latency constraints, which may require more powerful processing resources in terms of either CPU or RAM, see Tables 4.2 and 4.1. As we do not consider the placement of RU, we did not distinguish the RU energy requiremen

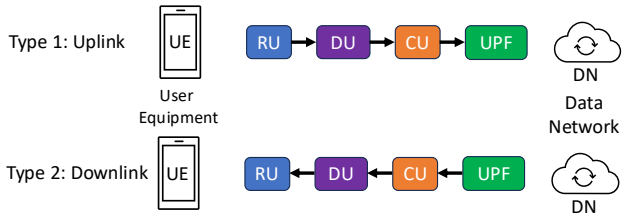


Figure 4.1: 5G E2E logical functionality chain (user plane)

Demand is then defined by a set of flow requests, denoted by K, where each flow request k is characterized by a source SRC_k, a destination DST_k, a bandwidth requirement b_k (Gbs), a delay requirement δ_{COS_k} and a logical functionality chain c_{COS_k}. Note that both E2E and processing delays depend on the class of service of k. In order to alleviate the notations, we will use δ_k and c_k for short from now on.

Two types of flows are considered: uplink flows, where the source is the RU and the destination is the Data Network (DN), and downlink flows, where the source is the data network and the destination is the RU. Each logical functionality chain (LFC) c is defined as an ordered sequence of 5G logical functionalities. We denote by f_i the ith logical functionality in any given logical functionality chain, where $i \in I_{\text{COS}_k} = \{1, \ldots, |c_k|\}$. Each logical functionality chain is mapped to a request depending on its direction (i.e., uplink or downlink). Let C be the set of all logical functionality chains.

Let F be the set of 5G logical functionalities, indexed by f. Each functionality f requires some compute resources, i.e., CPU, RAM and storage for its processing. Let $\text{CPU}_f, \text{RAM}_f, \text{STO}_f$ be the amount of CPU, RAM and storage required by f for each unit of bandwidth (it is not necessarily a linear function). Therefore, assuming f is one occurrence of the LFC functions in request k, $\text{VM}_{fk}^{\square}$ represents the fraction of $\square \in \{\text{RAM}, \text{CPU}, \text{STO}\}$ of processing pool resources (e.g., virtual machine (VM) or containers) (we assume that all processing pool resources have the same configuration) that it needs for each of its occurrences. Each logical functionality has a limited number of replicas, denoted by REPLICA_f.

Note that each VM can serve several demands or functionalities, with each demand/functionality only using a fraction of the compute resources of each VM. Note that the complexity of a functionality may not always scale linearly with the unit of bandwidth of the functionality.

The **Placement of 5G/B5G Logical Functionalities** problem is critical for efficiently provisioning 5G requests and making sure they can go through their required logical functionality chain. We formally state it as follows. For a given set of request flows, where each flow k is characterized by a 5-tuple, $(s_k, d_k, b_k, c_k, \delta_{\text{COS}_k})$, identify the best function locations in order to provision the set of LFCs and requests, while minimizing the overall network energy consumption subject to the transport and compute capacities (i.e., CAP_ℓ for the links and by $\text{CAP}^{\square}$ for $\square \in \{\text{CPU}, \text{RAM}, \text{STO}\}$ for the nodes), and optionally to a limit on the number of logical functionality replicas. When provisioning a request k, its required logical functionalities are encountered in the same order as in c_k, with some functions possibly located at the same node.

Table 4.1: Logical functionality chains (bandwidth and latency values adapted from [Askari et al., 2018])

Services	Class	Logical node sequences	Bandwidth per user or IoT system	E2E latency	Request bundles
Cloud gaming (CG)	eMBB	UPF_{CG} - CU - DU - RU	4 Mbps	80 ms	[40-55]
Augmented reality (AR)	eMBB	UPF_{AR} - CU - DU - RU	100 Mbps	10 ms	[1-4]
VoIP	eMBB	Uplink: RU - DU - CU - UPF_{VOIP} Downlink: UPF_{VOIP}- CU - DU - RU	64 Kbps	100 ms	[100-200]
Video streaming (VS)	eMBB	UPF_{VS} - CU - DU - RU	4 Mbps	100 ms	[50-100]
Massive IoT (MIOT)	mMTC	RU - DU - CU - UPF_{MIOT}	[1-50] Mbps	5 ms	[10-15]
Industry 4.0 (I4.0)	uRLLC	Uplink: RU - DU - CU - $UPF_{I4.0}$ Downlink: $UPF_{I4.0}$ - CU - DU - RU	70 Mbps	8 ms	[1-4]

Table 4.2: CPU/RAM/Storage/E^{BIT} usage for logical nodes

Logical nodes	vCPU	RAM (Gb) per 100 Mbps	Storage (Gb) per 100 Mbps	Logical node processing time per 0.01 msec unit	Energy per bit (E^{bit}) (pJ/bit)
RU	1	4	7	12	100
DU	9	5	1	6	80
CU	11	15	2	22	60
UPF_{CG}	13	15	7	14	30
UPF_{AR}	5	2	5	16	30
UPF_{VOIP}	5	2	5	2	30
UPF_{CG}	5	11	10	4	30
UPF_{MIOT}	5	3	20	4	30
$UPF_{I4.0}$	5	4	11	4	30

4.3.2 Layered Graph

In the context of Virtual/Container Network Function (VNF/CNF) placement, many authors have used the concept of layered graphs, see [Dwaraki and Wolf, 2016] for one of the early references, and then, e.g., [Huin et al., 2018; Hyodo et al., 2019]. Here, we propose a revisited layered graph adapted to the logical functionalities of 5G, and in particular, to certain location restrictions of logical nodes, i.e., DU and CU in the access network, while UPF is either in transport or in the core network, depending on latency requirements.

A layered graph G^{L} is defined for each request flow k and its associated logical functionality chain as follows. For an uplink (resp. downlink) flow chain RU $\rightarrow$ DU $\rightarrow$ CU $\rightarrow$ UPF (resp. UPF $\rightarrow$ CU $\rightarrow$ DU $\rightarrow$ RU), the layered graph has 3 layers, see Figure 4.3 for an illustration of the layered graphs for uplink and downlink request flows.

The initial network graph G is transformed into a *layered graph* G^{L} (not counting the RU_{SRC} (resp. DN_{SRC}) and RU_{DST} (resp. DN_{DST}) nodes), where each layer is either associated with the transport network graph (or a subgraph of it) or the combination of the transport and core networks. For every node $v \in V$,

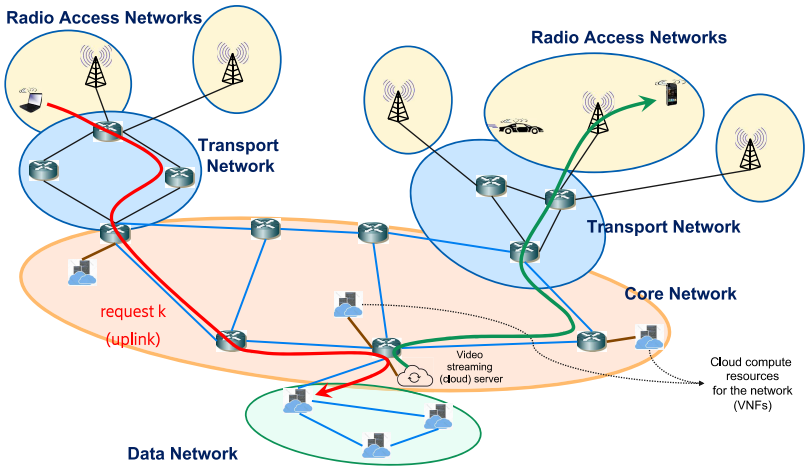
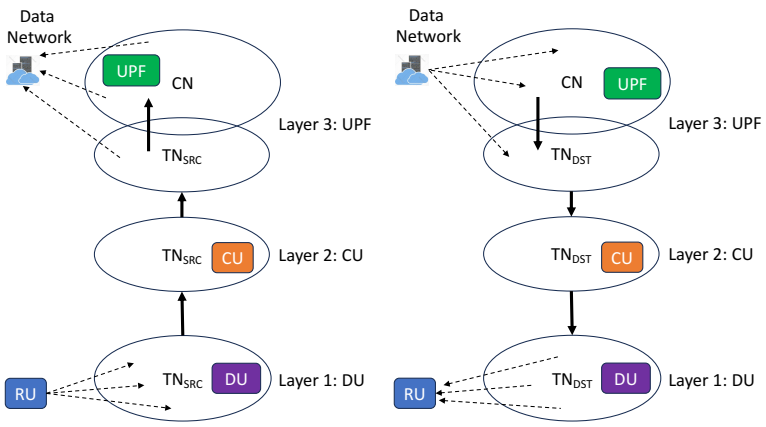


Figure 4.2: Overview of the various networks

let v^i be the corresponding node in the ith layer $i \in I_{\text{Cos}_k}$. Every $(i-1, i)$ layer pair is connected by cross-layer links from v^{i-1} to v^i, if the function f^c_{i-1} can be installed and run on node v, see Figure 4.4 for an illustration. Therein, request k first goes through its RU_{SRC} following the base station to which it connects, and then to the access network in order to go through a DU and then a CU. Next, it needs to identify a UPF, either in the transport or in the core network, and then again a CU and then a DU, before reaching a RU and then the destination. Note that we do not need to consider the overall transport network, e.g., in Figure 4.2, we only need to consider TN^{SRC}_k for the DU-UL/CU-UL layers, and similarly only TN^{DST}_k for the CU-DL/DU-DL layers. Finding a path and a chain placement for a request $(s_k, d_k, b_k, c_k, \delta_{\text{Cos}_k})$ consists in finding a path on the layered graph G^{L} from node RU_{SRC} to node RU_{DST}. Note that each layer represents the progression of the chain, e.g., being on the second layer means that the placement of the first function of the chain is already decided. The placement of the logical functionality associated with layer i is given by the cross-layer link used to switch between layers i and $i+1$.

4.3.3 Energy Model

In this section, we present the two key components of the energy model impacted by the placement of logical functionalities. These are: E_{PP}, the energy of the processing pools, i.e., the set of servers hosting the logical functionalities and processing the traffic, and E_{NET}, the energy of the network devices (e.g., routers,



(a) Uplink layered graph (b) Downlink layered graph

Figure 4.3: Layered graphs

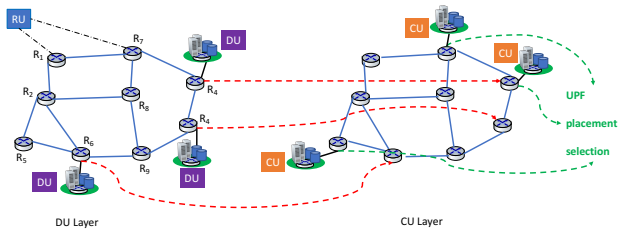


Figure 4.4: Details of E2E layered graph

optical fibers, and microwave links) that interconnect them. The energy model is then written:

$$E = E_{\text{NET}} + E_{\text{PP}}. \tag{4.1}$$

We then decompose each energy component into static and dynamic components. The static component corresponds to energy consumption outside of workload, i.e., when resources are unused. The dynamic component, on the other hand, is determined by resource usage due to workload.

Energy Model of the Communication Network

The energy consumption of the network linking processing pools among them and with the end users is presented in this section. As the placement of logical functionalities does not influence the energy consumed between the user and the RU, the energy of that part of the network is not considered in our model. For a

time period T, The energy of the communication network is:

$$E_{Net} = E_R + E_L, \tag{4.2}$$

where E_R and E_L are respectively the energy consumption of network nodes and network links respectively represented in Equations (4.3) and (4.4).

Energy Consumption of Network Nodes

A typical 5G network contain four different types of nodes: core, backbone, metro and feeder routers, all included in the set of routers R. The dynamic energy consumption of a router r relies on two factors: the energy needed for packet processing (E_r^{PKT}) and the energy required for storing and forwarding each byte of the payload (E_r^{SF}) [Ahvar et al., 2022]. The energy for packet processing remains constant regardless of packet size, as all packets necessitate processing. Conversely, the energy for storing and forwarding depends on packet length because each byte of the payload demands energy for reception, storage, switching across the fabric, and transmission over the link. We assume packets to have an average length of L bytes for the entire network. For a time period T, the energy consumption of the routers in the network can then be expressed as:

$$\begin{aligned} E_R &= \sum_{r \in R} \left(\text{Pow}_r^{\text{IDLE}} + E_r^{\text{PKT}} b^{\text{PKT}} + E_r^{\text{SF}} b^{\text{BYTE}}\right) T \\ &= \sum_{r \in R} \left(\text{Pow}_r^{\text{IDLE}} + \left(\frac{E_r^{\text{PKT}}}{8L} + \frac{E_r^{\text{SF}}}{8}\right) b^{\text{BIT}}\right) T, \end{aligned} \tag{4.3}$$

where $\text{Pow}_r^{\text{IDLE}}$ is the idle power consumption of the router r, b^{PKT}, b^{BYTE}, and b^{BIT} are the input packet, byte and bit rate respectively. Parameters $\text{Pow}_r^{\text{IDLE}}$, E_r^{PKT} and E_r^{SF} depend on the router type, i.e., its architecture and energy profile.

Energy Consumption of Network Links

Current 5G networks contain two types of links: fiber optical links and microwave links. For a time period T, the network link energy consumption can be expressed as:

$$E_L = \sum_{\ell \in L} (\text{Pow}_\ell^{\text{IDLE}}.T + E_\ell^{\text{DYN}}), \tag{4.4}$$

where $\text{Pow}_\ell^{\text{IDLE}}$ is the power consumed when link ℓ is powered on but not carrying data traffic, and its value

varies depending on whether it is optical or microwave. E_{ℓ}^{DYN} is the energy consumed by the network interfaces connecting the link while carrying data traffic. We will assume that this part of the energy is included within the incremental per-bit store and forward energy E^{SF} of the routers. Therefore, our focus will solely be on the static energy of the link.

Energy Model of the Processing Pools

Processing pools consist of physical machines (servers) that host logical functionalities implemented in virtual machines, containers. We do not differentiate between these implementation methods as specialized studies have been conducted on comparing the energy consumption of these virtual environments, see, e.g., [Al-Karawi et al., 2024], and also [Randal, 2020] for their pros and cons in practice.

The first energy component or processing pools is the static energy of its infrastructure, which includes the energy needed to maintain basic operating system processes and other idle tasks, such as the power required for leaking currents in semiconductor components like the CPU, memory, I/O, and other motherboard components. Additionally, there is the dynamic energy component, primarily influenced by the type of workload running on the infrastructure and how it utilizes the system's CPU, memory, I/O, and other resources. The energy consumed by other components of the data center, such as the cooling system, will be factored in by utilizing the power usage effectiveness (η^{PUE}) of the respective data center. For a time period T, the energy consumption of the processing pool centers can be expressed as:

$$E_{\text{PP}} = \sum_{v \in V^{\text{PP}}} \left(\text{Pow}_{v}^{\text{IDLE}} + \sum_{f \in F} E_{f}^{\text{BIT}} b_{v}^{\text{BIT}} \right) .T.\eta_{v}^{\text{PUE}}, \tag{4.5}$$

where $\text{Pow}_{v}^{\text{IDLE}}$ is the power consumed by a processing pool node v hosting one or several logical functionalities when carrying no data traffic, and E_{f}^{BIT} is the energy of function f to process 1 bit of data and b_{v}^{BIT} is the input bit rate of traffic on node v. Note that E_{f}^{BIT} depends on the traffic type, and η_{v}^{PUE} depends on the energy efficiency of the processing pool.

4.4 Proposed Column Generation and Scalable Heuristic Strategy

4.4.1 Proposed Column Generation

We propose a decomposition mathematical model, called LFP_CG model, which relies on the concept of configurations. Therein, a configuration is defined by a potential path provisioning, called *service path*, which is next described formally below for a given request flow.

A *Service Path* p for request flow k is a path going through the ordered logical functionalities of the service chain c_k associated with the class of service (CoS) of request k. Let P_k denote the set of service paths from SRC_k to DST_k.

Note that the same DU and CU must be shared by the set of request flows originating the same RU location.

Notations

- $V^{\text{C\_PP}}$ the candidate set of nodes to be connected to a processing pool node, able to run any logical functionality $f \in \{\text{DU}, \text{CU}, \text{UPF}\}$.
- V^{RU} is the set of RU nodes (we do not look at the placement of the RU functionality).
- K_{RU} is the set of requests originating from or destined to the UEs associated with RU. $K_{\text{RU}} = K_{\text{RU}}^{\text{UL}} \cap K_{\text{RU}}^{\text{DL}}$, where $K_{\text{RU}}^{\text{UL}}$ and $K_{\text{RU}}^{\text{DL}}$ are the set of requests originating and destined to the UEs associated with RU, respectively.

Parameters

- $a_{iv}^p \in \{0,1\}$, where $a_{iv}^p = 1$ if f_i^k is executed on node v for *Service Path* $p \in P_k$.
- $a_{i,\text{UL},v}^p$ (resp. $a_{i,\text{DL},v}^p$) $\in \{0,1\}$, where $a_{i,\text{UL},v}^p = 1$ (resp. $a_{i,\text{DL},v}^p = 1$) if $f_i^k = \text{DU}_{\text{UL}}$ or CU_{UL} (resp. $f_i^k = \text{DU}_{\text{DL}}$ or CU_{DL}) is executed on node v for *Service Path* $p \in P_k$.
- $\text{NUM}_\ell^p \in \mathbb{N}$ denotes the number of occurrences of link ℓ in path p: $\sum_{i \in I_k} \varphi_\ell^i$, where $\varphi_\ell^i = 1$ if path p goes through ℓ in layer i.
- $\text{NUM}_r^p \in \mathbb{N}$ denotes the number of occurrences of router r in path p: $\sum_{i \in I_k} \varphi_r^i$, where $\varphi_r^i = 1$ if path p goes through router r in layer i.

- $\text{PLACE}_{vf} \in \{0, 1\}$, where $\text{PLACE}_{vf} = 1$ if function f can be hosted on node v and 0 otherwise.

Variables

- $y_p^k \in \{0, 1\}$, where $y_p^k = 1$ if request k uses service path p, 0 otherwise.
- $z_{vf} \in \{0, 1\}$, where $z_{vf} = 1$ if function f is placed on node v and 0 otherwise.
- $z_v \in \{0, 1\}$, where $z_v = 1$ if v hosts at least one logical functionality $f \in F$ and 0 otherwise.
- $x^f_{\text{RU},v} \in \{0, 1\}$, where $x^f_{\text{RU},v} = 1$ if at least one request $k \in K^{\text{UL}}_{\text{RU}} \cup K^{\text{DL}}_{\text{RU}}$ executes function $f \in \{\text{DU}, \text{CU}\}$ on node v and 0 otherwise.
- $z_v^{\text{UPF}} \in \{0, 1\}$, where $z_v^{\text{UPF}} = 1$ if v hosts at least one $\text{UPF}_\circ$ logical functionality for a given service $\circ \in \{\text{CG}, \text{AR}, \text{VoIP}, \text{VS}, \text{MIoT}, \text{I4.0}\}$.

Objective

The objective is to minimize the overall energy consumption. Note that, as we assume that all links and routers are permanently active, there is no need to include the energy consumed during the idle state of links and routers, as it remains constant and independent of the solution. The objective is then expressed as:

$$\min \; E_{\text{NET}}(y_p^k) + E_{\text{PP}}(z_v, y_p^k), \tag{4.6}$$

where:

$$E_{\text{NET}}(y_p^k) = \sum_{r \in R} \sum_{k \in K} \sum_{p \in P_k} \left(\frac{E_r^{\text{PKT}}}{8L} + \frac{E_r^{\text{SF}}}{8} \right) \text{NUM}_r^p b_k T y_p^k, \tag{4.7}$$

$$E_{\text{PP}}(z_v, y_p^k) = \sum_{v \in V^{\text{C-PP}}} \text{Pow}_v^{\text{IDLE}} \eta_v^{\text{PUE}} T z_v + \sum_{v \in V^{\text{C-PP}}} \sum_{k \in K} \sum_{p \in P_k} \sum_{i \in I_k} E_{f_i k}^{\text{BIT}} b_k a_{iv}^p \eta_v^{\text{PUE}} T y_p^k. \tag{4.8}$$

Constraints

The set of constraints can then be expressed as follows.

One path per demand

$$\sum_{p \in P_k} y_p^k = 1, \qquad k \in K. \tag{4.9}$$

Link capacity

$$\sum_{k \in K} \sum_{p \in P_k} b_k \, \text{NUM}_\ell^p \, y_p^k \leq \text{CAP}_\ell, \qquad \ell \in L. \tag{4.10}$$

Processing pool node resource capacity

$$\sum_{k \in K} \sum_{i \in I_k} \sum_{p \in P_k} \text{VM}_{f_i k}^{\square} b_k \, a_{iv}^p \, y_p^k \leq \text{CAP}_v^{\square}, \qquad \square \in \{\text{CPU}, \text{RAM}, \text{STO}\}, v \in V^{\text{C\_PP}}. \tag{4.11}$$

Limited # of logical functionality occurrences

$$\sum_{v \in V^{\text{C\_PP}}} z_{vf} \leq \text{REPLICA}_f, \qquad f \in F. \tag{4.12}$$

Checking the existence of a service path, i.e., all its functions are hosted on a node along it

$$\sum_{p \in P_k} a_{iv}^p \, y_p^k \leq z_{vf_i}, \qquad v \in V^{\text{C\_PP}}, i \in I_k, k \in K. \tag{4.13}$$

Logical functionality placement

$$z_{vf} \leq \text{PLACE}_{vf}, \qquad f \in F, v \in V^{\text{C\_PP}}. \tag{4.14}$$

$$z_{vf} \leq z_v, \qquad f \in F, v \in V^{\text{C\_PP}}. \tag{4.15}$$

Limited number of UPF nodes

$$\sum_{v \in V^{\text{C\_PP}}} z_v^{\text{UPF}} \leq \text{UB}, \qquad f \in F^{\text{UPF}}. \tag{4.16}$$

$$z_{vf} \leq z_v^{\text{UPF}}, \qquad f \in F^{\text{UPF}}, v \in V^{\text{C\_PP}}. \tag{4.17}$$

For a given RU, outgoing and incoming flows must go through the same DU and CU

$$\sum_{k \in K_{\text{RU}}^{\text{UL}}} \sum_{p \in P_k} a^p_{f,\text{UL},v} y^k_p + \sum_{k \in K_{\text{RU}}^{\text{DL}}} \sum_{p \in P_k} a^p_{f,\text{DL},v} y^k_p \leq |K_{\text{RU}}| x^f_{\text{RU},v},$$

$$v \in V^{\text{C\_PP}}, \text{RU} \in V^{\text{RU}}, f \in \{\text{DU}, \text{CU}\}. \tag{4.18}$$

$$\sum_{v \in V^{\text{C\_PP}}} x^f_{\text{RU},v} = 1, \qquad f \in \{\text{DU}, \text{CU}\}, \text{RU} \in V^{\text{RU}}. \tag{4.19}$$

Solution of the Mathematical Model

The mathematical model of the previous section has an exponential number of variables and therefore requires a decomposition solution scheme using column generation techniques [Lübbecke and Desrosiers, 2005] in order to scale. The latter scheme reuses the mathematical model of the previous section as a so-called Master Problem (MP) and the Restricted Master Problem (RMP), i.e., MP with a very small subset of configurations/columns, and the so-called Pricing Problem (PP), i.e., a configuration generator. Consequently, the Restricted Master Problem corresponds to (4.6) - (4.19) with a very limited number of variables. Its role is to select the best provisioning, one for each request, while the pricing problem generates improving configurations, i.e., configurations such that, if added to the current RMP, improve the value of its linear relaxation. Once the linear relaxation of MP is solved, an integer solution is sought. Reader who is not familiar with column generation and how to seek an integer solution is referred to, e.g., [Chvatal, 1983] and [Barnhart et al., 1998]. Figure 4.5 provides an illustration of the flowchart of the solution when combining column generation and integer linear programming techniques.

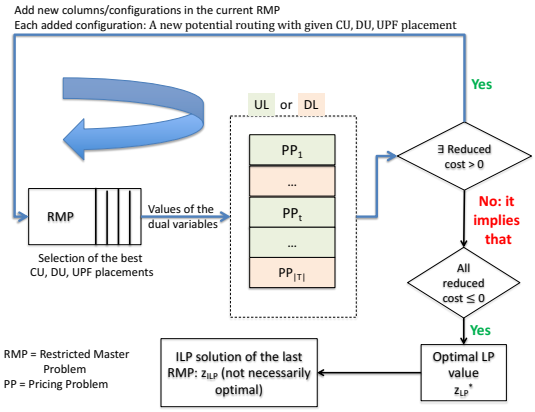


Figure 4.5: CG ILP flowchart

We now describe the pricing problem whose role is to generate a valid *Service Path* for a given request. Once again, the formulation relies on the layer graph (G^{L}) introduced in Section 4.3.2. Its objective is defined by the so-called reduced cost (see [Chvatal, 1983] if not familiar with linear programming concepts).

- $u^{(j)}$ represents the vector of dual variables of constraints (j) in the RMP.
- CONNECT_ℓ represents the set containing the two routers connecting link ℓ

Variables:

- $\alpha_{iv} \in \{0,1\}$, where $\alpha_{iv} = 1$ if f_i^k is installed on node v, 0 otherwise.
- φ_ℓ^i (resp. φ_r^i) $\in \{0,1\}$, where $\varphi_\ell^i = 1$ (resp. $\varphi_r^i = 1$) if the flow is forwarded on link ℓ (resp. router r) on layer i, i.e., links (resp. routers) in each layer in graph G^{L}, 0 otherwise.
- $\beta_{vf} \in \{0,1\}$, where $\beta_{vf} = 1$ if function f is installed in node v. Note that chains can contain multiple occurrences of the same function.

The service path generator (so-called pricing problem in mathematical programming and decomposition models) is written for each request k. We describe it below for the case of an uplink request, and it can be written in a similar way for the other case, i.e., for a downlink request. Its objective function, so-called reduced cost, can be expressed as follows.

$$\begin{aligned}\min \quad & b_k T \sum_{r\in R}\sum_{i=1}^{|c_k|}\left(\frac{E_r^{\text{PKT}}}{8L}+\frac{E_r^{\text{SF}}}{8}\right)\varphi_r^i + b_k T \sum_{v\in V^{\text{C\_PP}}}\eta_v^{\text{PUE}}\sum_{i=1}^{|c_k|}E_{f_i k}^{\text{BIT}}\alpha_{iv} - u_k^{(4.9)} + b_k\sum_{\ell\in L}u_\ell^{(4.10)}\sum_{i=1}^{|c_k|}\varphi_\ell^i \\ & + \sum_{\square\in\{\text{CPU,RAM,STO}\}} b_k \sum_{v\in V^{\text{C\_PP}}} u_{v,\square}^{(4.11)} \sum_{i=1}^{|c_k|}\text{VM}_{f_i^{c_k}}^{\square}\alpha_{iv} + \sum_{v\in V^{\text{C\_PP}}}\sum_{i=1}^{|c_k|}u_{k,v,i}^{(4.13)}\alpha_{iv} + \text{COST}_{\text{UL}} + \text{COST}_{\text{DL}}, \quad (4.20)\end{aligned}$$

where:

$\text{COST}_{\text{UL}} = \sum_{f\in\{\text{DU,CU}\}}\sum_{v\in V^{\text{C\_PP}}}\alpha_{f,\text{UL},v}u_{\text{RU}_k^{\text{UL}},f,v}^{(4.18)}$ for uplink requests and 0 for downlink requests.

$\text{COST}_{\text{DL}} = \sum_{f\in\{\text{DU,CU}\}}\sum_{v\in V^{\text{C\_PP}}}\alpha_{f,\text{DL},v}u_{\text{RU}_k^{\text{DL}},f,v}^{(4.18)}$ for downlink requests and 0 for uplink requests.

Constraints are written as follows.

Flow conservation: they correspond to flow constraints (i.e., route) from one logical functionality to the next one in the 5G E2E logical functionality chain for the $\text{SRC}_k \rightsquigarrow \text{DST}_k$ request for which the pricing

problem is solved (constraints (4.22)), and then flow constraints from the source node to the location of the DU function of the service chain (constraints (4.21)), and similarly from the location of the last function, i.e., UPF for an uplink request, of the service chain to the destination node (constraints (4.23)). Note that $a_v^i = 0$ for all nodes that do not host any logical node/functionality and that we take care of the possibility that several logical functionalities can be located on the same node, including on the source or destination nodes.

Let $k \in K^{\text{UL}}$. Denote by TN_k the transport network associated with k as in Figure 4.2.

Selecting the DU placement

$$\sum_{\ell \in \omega^+(v)} \varphi_\ell^{\text{TN}_k^{\text{DU}}} - \sum_{\ell \in \omega^-(v)} \varphi_\ell^{\text{TN}_k^{\text{DU}}} + \alpha_{\text{DU,UL},v} = \begin{cases} 1 \text{ if } v = \text{SRC}_k \\ 0 \text{ else} \end{cases}, \quad v \in \text{TN}_k^{\text{DU}}. \tag{4.21}$$

Selecting the CU placement

$$\sum_{\ell \in \omega^+(v)} \varphi_\ell^{\text{TN}_k^{\text{CU}}} - \sum_{\ell \in \omega^-(v)} \varphi_\ell^{\text{TN}_k^{\text{CU}}} + \alpha_{\text{CU,UL},v} - \alpha_{\text{DU,UL},v} = 0, \quad v \in V^{\text{TN}_k^{\text{CU}}}. \tag{4.22}$$

Selecting the UPF placement

$$\sum_{\ell \in \omega^+(v)} \varphi_\ell^{\text{TN}_k^{\text{UPF}} \cup \text{CN}} - \sum_{\ell \in \omega^-(v)} \varphi_\ell^{\text{TN}_k^{\text{UPF}} \cup \text{CN}} - \alpha_{\text{UPF},v} = \begin{cases} -1 \text{ if } v = \text{DST}_k \\ 0 \text{ else} \end{cases}, \quad v \in \text{TN}_k^{\text{UPF}} \cup \text{CN}. \tag{4.23}$$

Link capacity.

$$b_k \sum_{i=0}^{|c_k|} \varphi_\ell^i \leq \text{CAP}_\ell, \qquad \ell \in L. \tag{4.24}$$

Compute node capacity. For $v \in V^{\text{LN}}$,

$$b_k \sum_{i=0}^{|c_k|-1} (\text{VM}^\square{}_{f_i^{c_k}}) \times \alpha_{iv} \leq \text{CAP}_v^{\text{VM}^\square}, \quad \text{VM}^\square \in \{\text{CPU}, \text{RAM}, \text{STO}\}. \tag{4.25}$$

Delay constraint for request k: it includes a link delay (propagation and average queuing delays: it corresponds to the contribution of k to the overall queuing delay on link ℓ, with the assumption that queuing delay depends linearly on the bandwidth , and a node delay (function processing delay).

$$\sum_{\ell \in L} \sum_{i=0}^{|c_k|} \delta_{\ell,k} \varphi_\ell^i + \sum_{v \in V^{\text{LN}}} \sum_{i=0}^{|c_k|-1} \delta_{v,f_i^{c_k}} \alpha_{iv} \leq \delta_k. \tag{4.26}$$

Routers connecting a link constraint: If a link ℓ is used on layer i, then so are the routers connecting that link.

$$\varphi_\ell^i \leq \varphi_r^i \qquad \ell \in L, \quad r \in \text{CONNECT}_\ell, i \in \{1, \ldots, |c_k| - 1\}. \tag{4.27}$$

4.4.2 Proposed Heuristic

In order to speed up model computation and improve scalability to larger networks, we proposed a heuristic algorithm (LFP_H) to determine the placement of DU, CU and UPF and the routing of request flow, ensuring that latency and capacity constraints are respected. Our heuristic leverages some features of the exchange algorithm [Zhang et al., 2005] which was originally developed for the facility location problem. As illustrated in Figure 4.6, the heuristic LFP_H presented in Algorithm 4.1 follows a hierarchical structure, organized into layers, as outlined in Section 4.3.2, to find the optimal placement.

The process begins by generating an energy profile for the candidate DU placement based on the traffic from the RUs. This energy profile is derived by identifying the best routing paths connecting RUs to candidate DU sites (see LFP_FEATURES described in Algorithm 4.3). Next, the energy profile is used as input feature for a clustering algorithm (e.g., Kmeans) that groups RUs and maps each cluster to a candidate processing pool (PP) of DU. This mapping provides the energy consumption associated with connecting a specific RU cluster to a given DU candidate. An initial solution is then generated by connecting each RU cluster to the DU candidate that minimizes energy consumption (see Algorithm 4.4). Following this, the initial solution is refined using an exchange-based heuristic. The exchange algorithm iteratively swaps nodes in the solution to improve the overall placement (see LFP_EXCHANGE described in Algorithm 4.5). Once the solution is optimized for DU placement, the process is repeated for the CU and UPF placements. After the DU, CU, and UPF placements are determined, all requests are provisioned based on these placements, ensuring that all capacity and delay constraints are satisfied. The total energy consumption for this

Table 4.3: Heuristic parameters definition

Parameter	Description
$cluster^f$	Mapping to each node $w \in \hat{V}^{f^-}$ to a node $v \in V^{\text{PP}}$ such that f^- hosted in w is connected to f hosted in v
$cluster^f_w$	Node $v \in V^{\text{PP}}$ such that f^- hosted in w is connected to f hosted in v
E^f	Array containing energy consumed per node pair to connect function f^- to function f
E^f_{wv}	Energy consumed to connect f^- hosted on node w to f hosted on node v
f^-	Function at layer $i-1$
f	Function at layer i
K^f	Set of request destined to function f
$path^f$	List of paths selected to connect function f^- to function f
$V^{\text{C\_PP}}$	Candidate set of processing pool nodes, i.e., of nodes with resources for executing at least one logical functionality
V^{PP}	Set of nodes selected to host a logical functionality f
$\hat{V}^{f^-}$	Set of nodes selected to host logical functionality f^-
$V^f_{notTried}$	Set of nodes in $V^{\text{C\_PP}}$ not yet chosen in the exchange argument process to improve the solution

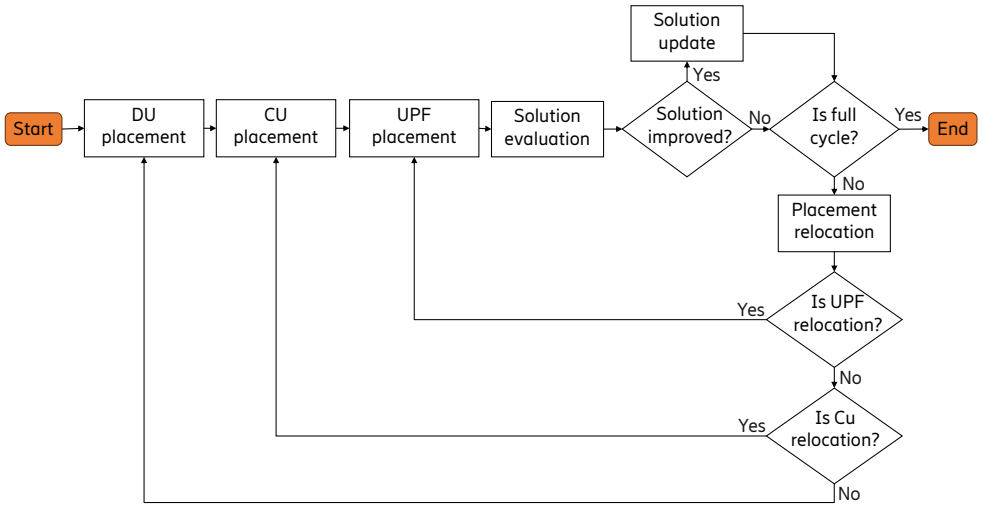


Figure 4.6: LFP_H heuristic flowchart

configuration is then calculated (see Algorithm 4.2). At this stage, a final exchange-based optimization is applied to explore the possibility of relocating DU, CU, or UPF nodes to different sites that are not currently included in the solution. This step involves generating new routing paths that satisfy the latency and capacity constraints, further enhancing the solution. The relocation and routing steps continue until no further improvements can be made.

In the following sections, we delve deeper into the key components of the proposed LFP_H and explain the methodology in more detail. The parameters used in the heuristic algorithm are described in Table 4.3

Algorithm 4.1 LFP_H: Heuristic solution

procedure LFP_H(G, K_{DU}, V^f for all $f \in F$)
  $G^{\text{L}} \leftarrow$ layered graph as described in Section 4.3.2
  $E^{\text{DU}}, path^{\text{DU}} \leftarrow$ LFP_FEATURES($G^{\text{L}}, DU, K_{\text{DU}}$)
  $\hat{V}^{\text{DU}}, cluster^{\text{DU}} \leftarrow$ LFP_INIT_PLACEMENT($E^{\text{DU}}, V^{\text{DU}}$)
  $\hat{V}^{\text{DU}}, cluster^{\text{DU}} \leftarrow$ LFP_EXCHANGE($E^{\text{DU}}, V^{\text{DU}}, \hat{V}^{\text{DU}}, V^{RU}$)
  $V^{\text{DU}}_{notTried} = V^{\text{DU}} \setminus \hat{V}^{\text{DU}}$
  $K_{\text{CU}} \leftarrow$ aggregate K_{DU} based on $\hat{V}^{\text{DU}}$ and $cluster^{\text{DU}}$
  $E^{\text{CU}}, path^{\text{CU}} \leftarrow$ LFP_FEATURES($G^{\text{L}}, CU, K_{\text{CU}}$)
  $\hat{V}^{\text{CU}}, cluster^{\text{CU}} \leftarrow$ LFP_INIT_PLACEMENT($E^{\text{CU}}, V^{\text{CU}}$)
  $\hat{V}^{\text{CU}}, cluster^{\text{CU}} \leftarrow$ LFP_EXCHANGE($E^{\text{CU}}, V^{\text{CU}}, \hat{V}^{\text{CU}}, \hat{V}^{\text{DU}}$)
  $V^{\text{CU}}_{notTried} = V^{\text{CU}} \setminus \hat{V}^{\text{CU}}$
  for UPF in UPFset **do**
    $K_{\text{UPF}} \leftarrow$ aggregate subset of K_{CU} of application upf
    based on $\hat{V}^{\text{CU}}$ and $cluster^{\text{CU}}$
    $E^{\text{UPF}}, path^{\text{UPF}} \leftarrow$ LFP_FEATURES($G^{\text{L}}, upf, K_{\text{UPF}}$)
    $\hat{V}^{\text{UPF}}, cluster^{\text{UPF}} \leftarrow$ LFP_INIT_PLACEMENT($E^{\text{UPF}}, V^{\text{UPF}}$)
    $\hat{V}^{\text{UPF}}, cluster^{\text{UPF}} \leftarrow$ LFP_EXCHANGE($E^{\text{UPF}}, V^{\text{UPF}}, \hat{V}^{\text{UPF}}, \hat{V}^{\text{CU}}$)
    $V^{\text{UPF}}_{notTried} = V^{\text{UPF}} \setminus \hat{V}^{\text{UPF}}$
  end for
  Obj $\leftarrow$ LFP_OBJECTIVE($G^{\text{L}}, K_{\text{DU}}$)
  $V_{notTried} = \cup_{f \in F} V^{\text{PP}}_{notTried}$
  while $V_{notTried} \neq \emptyset$ **do**
    $\bar{f} \leftarrow$ select a function $f \in F$ where $V^{\text{PP}}_{notTried} \neq \emptyset$
    $u \leftarrow$ select a node in $V^{\bar{f}}_{notTried}$
    for $v \in \hat{V}^{\bar{f}}$ **do**
      $\Delta E_v \leftarrow \sum_{w \in V^{\bar{f}_{i-1}}} E_{wv} - E_{wu}$
    end for
    $\bar{v} \leftarrow v \in \hat{V}^{\bar{f}} : \Delta E_{\bar{v}} < 0, \Delta E_{\bar{v}} = \min_{v \in \hat{V}^{\bar{f}}} \Delta E_v$
    if node $\bar{v}$ can be found **then**
      $\hat{V}_{oldsolution} \leftarrow$ save the current placement and
      routings for all function $f \in F$
      $\hat{V}^{\bar{f}} \leftarrow \hat{V}^{\bar{f}} \cup \{u\} \setminus \{\bar{v}\}$
      $cluster^{\bar{f}} \leftarrow$ update cluster with new $\hat{V}^{\bar{f}}$
      for all functions f after $\bar{f}$ **do**
        $K_f \leftarrow$ aggregate K_f based on $\hat{V}^{f^-}$
        and $cluster^{f^-}$
        $E^f, path^f \leftarrow$ LFP_FEATURES(G^{L}, f, K_{f^-})
        $V^{\text{PP}}, cluster^f \leftarrow$ LFP_INIT_PLACEMENT(E^f, V^{PP})
        $V^{\text{PP}}, cluster^f \leftarrow$ LFP_EXCHANGE($E^f, V^{\text{C\_PP}}, V^{\text{PP}}, \hat{V}^{f^-}$)
      end for
      New_obj $\leftarrow$ LFP_OBJECTIVE($G^{\text{L}}, K_{\text{DU}}$)
      if New_obj ¡ Obj **then**
        Obj $\leftarrow$ New_obj
      else
        $V^{\text{PP}} = V^{\text{PP}}_{oldsolution}$ for all function $f \in F$
      end if
    end if
    $V^{\hat{f}}_{notTried} \leftarrow V^{\hat{f}}_{notTried} \setminus \{u\}$
    $V_{notTried} \leftarrow V_{notTried} \setminus \{u\}$
  end while
  return V^{PP}
end procedure

Algorithm 4.2 LFP_OBJECTIVE: Get objective function

```
1: procedure LFP_OBJECTIVE(LG, K_DU)
2:     Obj ← 0
3:     for k ∈ K_DU do
4:         path_k ← ∅
5:         for f ∈ c_k do
6:             path_k ← path_k ∪ path_k^f
7:         end for
8:         latency_k ← compute latency of path_k
9:         if latency_k ≤ δ_k then
10:            energy_k ← get energy of path_k computed as
11:            in Section 4.3.3
12:            Obj ← Obj + energy_k
13:        else
14:            Obj ← ∞
15:            break
16:        end if
17:    end for
18:    return Obj
19: end procedure
```

Getting the Initial Placement a given Layer i

Given the traffic of nodes in layer $i-1$ for function f^-, this step generates the initial placement by successively running Algorithm 4.3 to extract the input features needed to connect function f^- in layer $i-1$ to function f in layer i, followed by Algorithm 4.4 to determine the initial placement of function f in layer i.

Energy features of LFP_FEATURES

LFP_FEATURES represented in Algorithm 4.3 finds the routing from function f^- in layer $i-1$ to function f in layer i, taking as input the layered graph LG, the function f, and the set of requests destined to f. This is done by searching for the routing path that minimizes the energy for each pair (w, v), where $w \in \hat{V}^{f^-}$ is the set of selected PP to host f^- in layer $i-1$, and $v \in V^{\text{C\_PP}}$ is the set of candidate nodes to host f in layer i. The needed energy to provision requests in these paths while satisfying latency and delay constraints are then compute. The output of this algorithm consists of energy features that will be used to create the initial solution.

Getting the placement with LFP_INIT_PLACEMENT

LFP_INIT_PLACEMENT represented in Algorithm 4.4 takes as input the energy feature E^f, generated by Algorithm 4.3, the set V^f of candidate sites to host function f, the selected nodes $\hat{V}^{f^-}$ to host function f^-, and returns the selected nodes V^{PP} as well as the connectivity between the selected nodes to process f^- and the selected nodes to host f ($cluster^f$). This algorithm uses a clustering technique to group the selected nodes for hosting f^- and assigns to each cluster the candidate node for hosting f that minimizes the energy.

Algorithm 4.3 LFP_FEATURES: Energy feature generation from function f^- to function f

procedure LFP_FEATURES(G^{L}, f, K^f)
  $E^f \leftarrow$ empty array of size $|\hat{V}^{f^-}| \times |V^f|$
  $paths^f \leftarrow \emptyset$
  for $w \in \hat{V}^{f^-}$ **do**
    for $v \in V^{\text{C\_PP}}$ **do**
      $P \leftarrow$ candidate paths connecting w and v with
      available capacity
      $energy \leftarrow 0$
                                ▷ Assign paths to each request flow originating from w
      while $K_w^f \neq \emptyset$ and $P \neq \emptyset$ **do**
        $p \leftarrow$ shortest path in P with available capacity
        $cap_p \leftarrow$ available capacity on p
        $subK \leftarrow$ subset of K_w^f s.t $\sum_{k \in subk} B_k \leq cap_p$
        if $subK \neq \emptyset$ **then**
          save p in $paths^f$ for $k \in subK$
          $e \leftarrow$ get energy computed as in Section 4.3.3
          $energy \leftarrow energy + e$
        end if
        $K_w^f \leftarrow K_w^f \setminus subK$
        $P \leftarrow P \setminus \{p\}$
        update network state
      end while
      if $K_w^f = \emptyset$ **then** ▷ Available paths for all $k \in K_w^f$
        $E_{wv}^f \leftarrow energy$
      else
        $E_{wv}^f \leftarrow \infty$ ▷ No available paths available for all $k \in K_w^f$
      end if
    end for
  end for
  return $E^f, \quad paths^f$
end procedure

Algorithm 4.4 LFP_INIT_PLACEMENT: Generate the initial placement of function f

procedure LFP_INIT_PLACEMENT($E^f, V^{\text{PP}}, \hat{V}^{f^-}$, REPLICA$_f$)
  $V^{\text{PP}} \leftarrow \emptyset, \quad cluster^f \leftarrow$ empty array of size $|\hat{V}^{f^-}|$
  $clusters \leftarrow$ run a clustering algrotihm using E^f features
  generate REPLICA$_f$ clusters of node $w \in \hat{V}^{f^-}$
  for $cluster \in clusters$ **do**
    $\bar{v} \leftarrow$ find node $v \in V^{\text{C\_PP}}$ such that $\sum_{w \in cluster} E_{wv}^f$ is
    minimum
    $V^{\text{PP}} \leftarrow V^{\text{PP}} \cup \{\bar{v}\}$
    for $w \in cluster$ **do**
      $cluster_w^f \leftarrow \bar{v}$
    end for
  end for
  return $V^{\text{PP}}, cluster^f$
end procedure

Algorithm 4.5 LFP_EXCHANGE: Improve placement of function $f \in F$ in layer i

procedure LFP_EXCHANGE($E^f, V^{\text{C\_PP}}, V^{\text{PP}}, \hat{V}^{f^-}$)
  $V^f_{notTried} = V^{\text{C\_PP}} \setminus V^{\text{PP}}$
  while $V^f_{notTried} \neq \emptyset$ **do**
    $u \leftarrow$ select a node in $V^f_{notTried}$
    for $v \in V^{\text{C\_PP}}$ **do**
      $\Delta E^f_v \leftarrow \sum_{w \in \hat{V}^{f^-}} (E^f_{wv} - E^f_{wu})$
    end for
    ▷ Select the node that can be replaced by u
    $\bar{v} \leftarrow \{v \in V^{\text{C\_PP}} : \Delta E^f_{\bar{v}} < 0, \Delta E^f_{\bar{v}} = \min_{v \in V^{\text{C\_PP}}} \Delta E^f_v\}$
    if node $\bar{v}$ can be found **then**
      $V^{\text{PP}} \leftarrow V^{\text{PP}} \cup \{u\} \setminus \{\bar{v}\}$ ▷ Substitute $\bar{v}$ by u
    end if
    $V^f_{notTried} = V^f_{notTried} \setminus \{u\}$
  end while
  $cluster^f \leftarrow$ empty array of size $|\hat{V}^{f^-}|$
  for $w \in \hat{V}^{f^-}$ **do**
    $cluster^f_w \leftarrow$ get node $v \in V^{\text{PP}}$ s.t $E^f_{wv} = min_{v \in V^{\text{PP}}} E^f_{wv}$
  end for
  return $V^{\text{PP}}, cluster^f$
end procedure

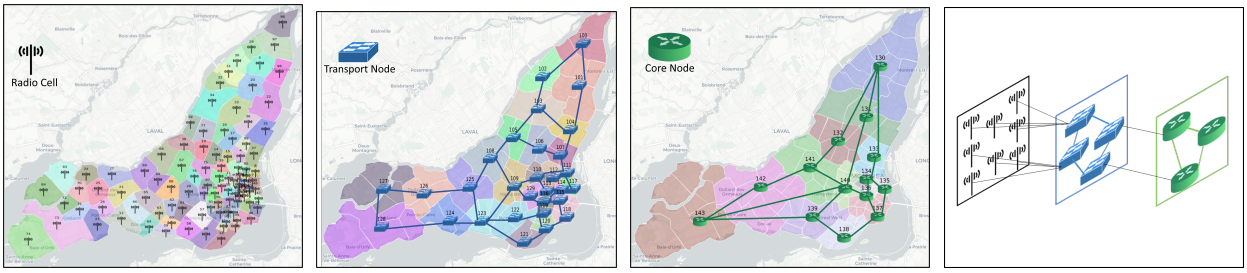
Improving solution with exchange argument LFP_EXCHANGE

The proposed exchange algorithm LFP_EXCHANGE, described in Algorithm 4.5, is a heuristic approach used here to iteratively improve solutions through the exchange of PP nodes, thereby balancing computational efficiency with solution quality. While it does not guarantee an optimal solution, it provides an effective and scalable method for large-scale PP node selection. For each function f, the initial solution found with Algorithm 4.4 is provided as input, along with the energy feature E^f, the set $V^{\text{C\_PP}}$ of candidate sites to host logical functionalities, and the selected nodes to host function f^- ($\hat{V}^{f^-}$). The objective of this algorithm is to generate an improved solution V^{PP} by swapping nodes in the current solution with nodes from the candidate set. If it finds candidate nodes in $V^{\text{C\_PP}}$ that are not yet in V^{PP} and would significantly improve the current solution, the substitution is performed, and V^{PP} is updated accordingly. Nodes in $\hat{V}^{f^-}$ are then reassigned to the nodes in the updated V^{PP} to generate the new mapping, $cluster_f$. LFP_EXCHANGE is then generalized and integrated into LFP_H to enhance the overall end-to-end (E2E) solution.

4.5 Numeral Results

4.5.1 Benchmark Algorithms

We evaluated the performance of our proposed algorithms (LFP_CG and LFP_H) by comparing them against two benchmark algorithms, outlined as follows:



(a) Radio Network (b) Transport network (c) Core network (d) Network hierarchy

Figure 4.7: 5G network where the placement will be done: radio, transport and core infrastructures and their hierarchy

- **Baseline 1**: A distance-based algorithm that places logical functions on the closest node that satisfies both capacity and delay constraints. Specifically, each RU is connected to the nearest candidate Physical Node (PP) running the DU. Each DU is connected to the closest candidate PP running the CU, and each CU is connected to the nearest candidate running the UPF. All placements and routings are made in compliance with the constraints defined in Section 4.4.1.

- **Baseline 2**: A greedy energy-based algorithm that places logical functions on PP candidates in a way that minimizes energy consumption. For each node (RU, DU, CU, respectively), it establishes a connection to another node running the subsequent function (DU, CU, UPF, respectively) only if that node is the candidate PP that minimizes energy consumption for the connection. The placement and routings are made while adhering to all the constraints specified in Section 4.4.1.

4.5.2 Experimental Setup and Dataset

All experiments were performed on a computer featuring an Intel® Xeon® Gold 620R CPU running at 2.40 GHz, equipped with 16 GB of RAM, and executed within a Python environment utilizing the GUROBI optimization library[1].

We used the data generated in Chapter 3. It contains E2E traffic generated by refactoring the urban data of the city of Montreal. The 5G cell locations are inspired by the real cell locations of a mobile operator in the city of Montreal, with an aggregated total of 100 RU nodes. 5G transport (30 nodes) and core nodes (15 nodes) are mapped by aggregating cells in the same neighbourhood, and network connectivity is done as in Figure 4.7. The algorithms will select nodes where we can place processing pools for logical

[1]https://www.gurobi.com/

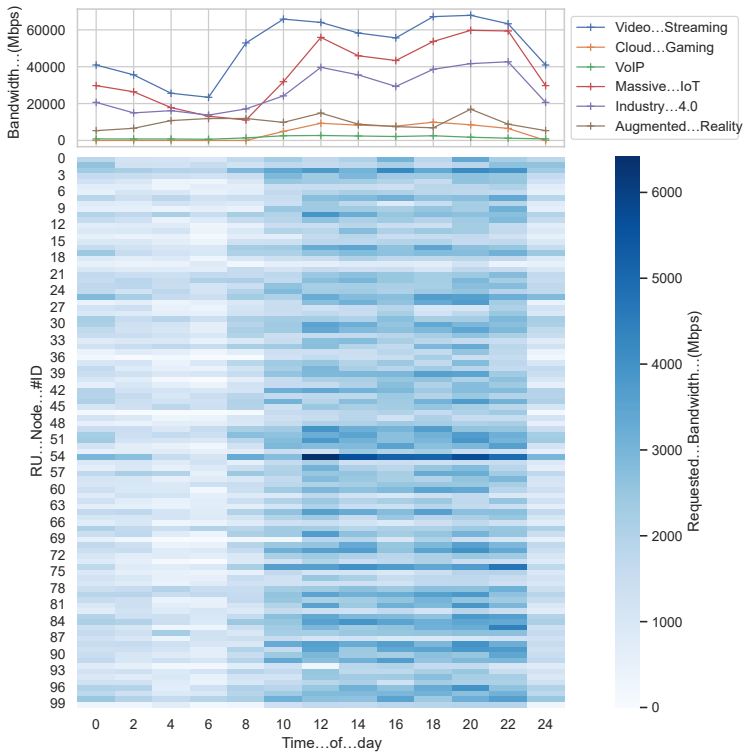


Figure 4.8: Aggregated input bandwidth per RU over a 24-hour period for each service class

functionalities (DU/CU/UPF). We considered six types of services, each with distinct bandwidth and delay requirements, as shown in Table 4.1. Each type of service is characterized by a distinct traffic profile, as shown in Figure 4.8 with varying peak types and traffic behaviors. The temporal and spatial distribution of the requested bandwidth for each RU node is also depicted in Figure 4.8 highlighting periods of peak traffic and areas (i.e., RU) with high, medium, or low traffic over a 24-hour period.

Table 4.2 outlines the modeling of compute resources and energy consumption per bit for each logical node. This includes the required CPU (as a percentage of CPU per user), RAM, and storage for each logical node, as well as the typical energy per bit consumed by the logical nodes (RU, DU, CU), and UPF). As we move closer to the core of the network, the energy consumption per bit is determined by the tasks performed by each logical functionality. The RU handles radio frequency signals and is set to consume the highest energy per bit, because of its high-power RF components. The DU handles lower-layer protocol processing, such as scheduling, data encoding/decoding, and resource allocation, resulting in moderate energy bit per compared to the RU. The CU responsible for higher-level network functions, including mobility management and policy control, has a lower energy per bit than the RU and DU. Finally,

the UPF focusing primarily on data forwarding, typically has a moderate energy per bit compared to the other units.

We assume that the processing pools in the transport network and core network differ only in size, with the processing pool in the core being larger than that in the transport network. However, both pools share similar hardware types and cooling systems. We define the typical idle power of a server as $\text{Pow}^{\text{IDLE}}_{server} = 50W$, and the idle power $\text{Pow}^{\text{IDLE}}_{v}$ of a node v is proportional to the number of active servers, which in turn depends on the traffic handled by the node. Additionally, we assume a power usage effectiveness η^{PUE}_{v} of 1.5 for all nodes $v \in V$, with the assumption that the majority of the energy consumption is attributed to the IT equipment, while only a small fraction is consumed by the non-IT infrastructure. These values do not come from real use cases (due to lack of access to real data) and only provide a certain order of magnitude. More details about the data set can be found in [Jaumard and Ziazet, 2023]. To compute the energy consumption of the network, we used the energy parameters of the routers r, i.e., $\text{Pow}^{\text{IDLE}}_{r}$, E^{PKT}_{r}, E^{SF}_{r}, and L based on the values provided in [Vishwanath et al., 2014].

4.5.3 Effectiveness of the Proposed Models LFP_CG and LFP_H

Since traffic distribution varies both spatially (from one RU to the next) and temporally (from one hour to the next), we evaluate the performance of each approach by determining the optimal placement for every hour and analyzing how each approach performs.

Placement analysis

Figure 4.9 illustrates the placement of DU, CU, and UPF for each approach across a 24-hour period. As shown, the proposed methods, LFP_CG and LFP_H, dynamically adjust the placement at each hour to optimize for changing traffic patterns. This adaptability enables the models to continuously update the placement solution in response to new traffic conditions. Consequently, as seen in Figure 4.9, the selected nodes for placement vary significantly from one hour to the next, ensuring the best possible solution based on the current traffic demands.

In contrast, the baseline solutions, which are rule-based, exhibit a more static placement strategy, where the same nodes are consistently selected throughout the day. This highlights the baseline's inability to effectively adapt to the temporal and spatial variations in traffic. As a result, the baseline approaches lead to

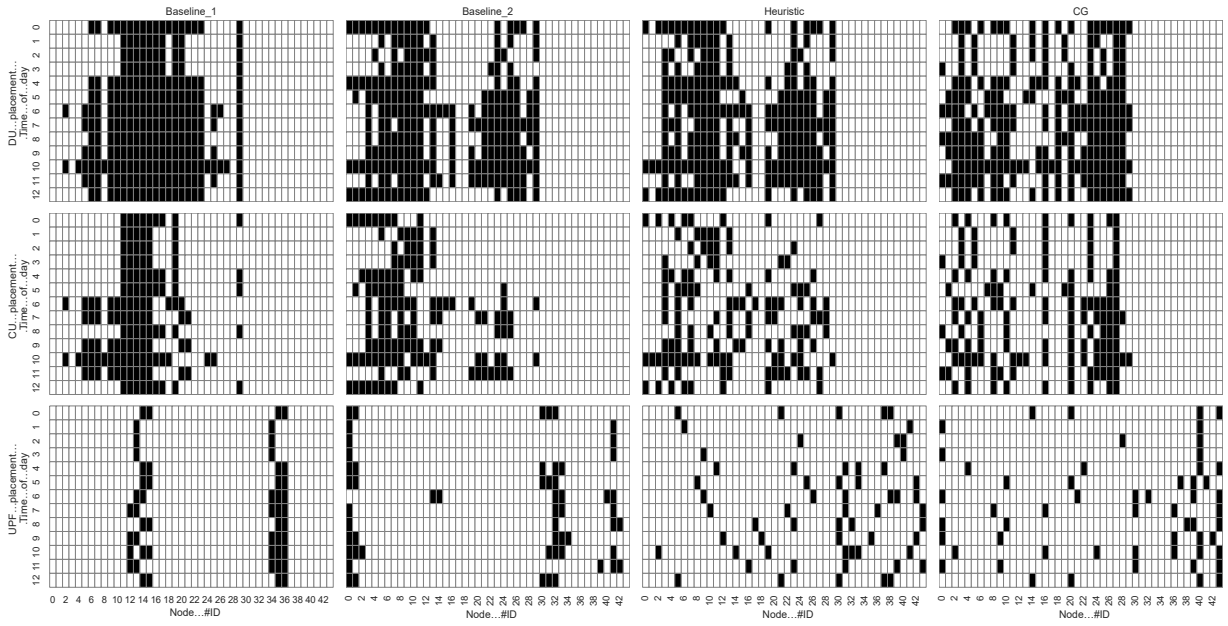


Figure 4.9: Node selection for placing logical functionalities over time: Black indicates the selected nodes. The plot shows that baseline selections are denser, with placements remaining relatively consistent from one hour to the next. In contrast, the proposed methods exhibit a sparser selection pattern, where placements can vary significantly between consecutive hours.

suboptimal performance and higher energy consumption, as will be further discussed in the next section.

Energy analysis

An important consideration when optimizing placement to minimize energy consumption is understanding that the total energy consumption can be divided into two main components: placement-dependent energy consumption and placement-independent energy consumption. Placement-dependent energy consumption refers to the energy that varies based on the DU/CU/UPF placement. This includes the energy used by the network to route traffic demands and the idle power consumed by the processing pool nodes. Since the placement impacts routing, the energy consumption for network operations is influenced, and each selected processing pool (PP) node will consume idle power. In contrast, placement-independent energy consumption remains constant, unaffected by the placement strategy or algorithm. It represents the energy required to process the demands at each node, assuming all nodes have the same hardware, as described in Section 4.5.2.

Figure 4.10 illustrates the energy consumption of each approach over a 24-hour period. As observed, both LFP_CG and LFP_H outperform the baseline approaches. LFP_H, in particular, performs exceptionally

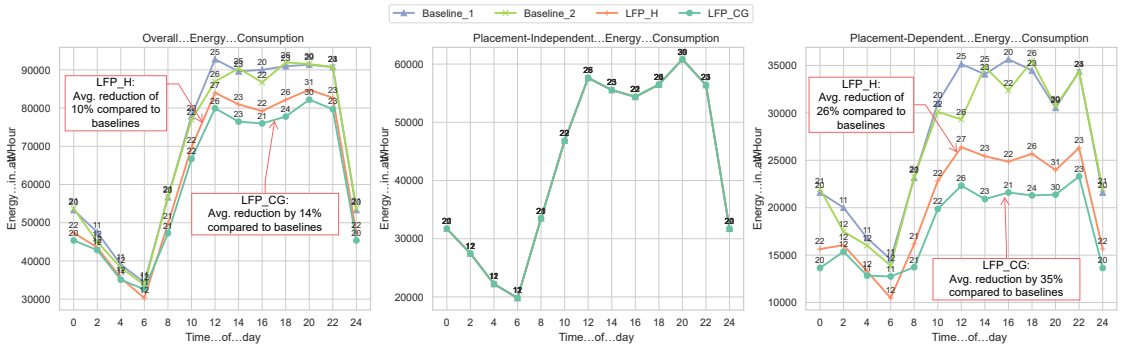


Figure 4.10: Energy comparison of proposed logical functionality placement (LFP) methods (**LFP_H**, **LFP_CG**) and baselines: Placement-independent energy consumption refers to the constant energy consumption that remains unaffected by DU/CU/UPF placement. Placement-dependent energy consumption represents the energy that varies based on the DU/CU/UPF placement. The numbers on the plot indicate the number of nodes selected for placing DU, CU, and UPF at each time of day for each method.

well during low traffic periods, occasionally even surpassing LFP_CG. During high traffic periods, LFP_H's performance aligns more closely with that of LFP_CG. When considering the overall energy consumption, LFP_H achieves a 10% reduction in energy compared to the baselines, while LFP_CG achieves a 14% reduction over the 24-hour period. These gains are more pronounced when focusing on the energy portion that is most affected by placement, i.e., the placement-dependent energy. In this case, LFP_H achieves a 26% reduction, while LFP_CG results in a 35% reduction compared to the baselines. This significant improvement is due to our proposed solutions' ability to dynamically adapt to the changing network profile by adjusting the placement of logical functionalities and routing.

4.5.4 Running Time vs Accuracy

We now evaluate the performance of each solution technique, focusing on the tradeoff between accuracy and running time. The optimal value of the linear programming relaxation (z_{LP}) of LFP_CG provides a lower bound on the optimal value. This value is then used to calculate the optimality gap (ε_{SOL}) of each solution technique (denoted by SOL) as follows:

$$\varepsilon_{\text{SOL}} = \frac{z_{\text{SOL}} - z_{\text{LP}}}{z_{\text{SOL}}},$$

where z_{SOL} represents the value of the objective function for the solution obtained by solution technique SOL.

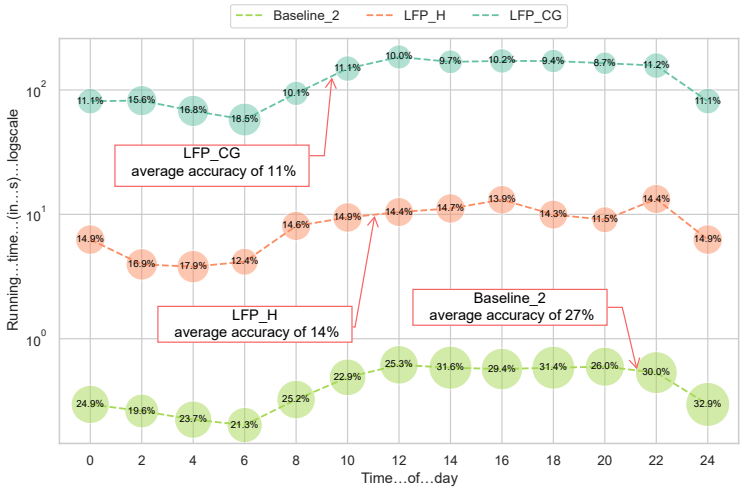


Figure 4.11: Time comparison of solutions (**LFP_H**, **LFP_CG**) and baseline 2: The numbers on the plot represent the optimality gap referred to as accuracy of solution SOL, calculated as $\varepsilon_{\text{SOL}} = \frac{z_{\text{SOL}} - z_{\text{LP}}}{z_{\text{SOL}}}$. A lower value indicates better accuracy.

Figure 4.11 shows the running time of the proposed model in comparison to the best baseline (baseline2) under time-varying traffic conditions. It also displays the accuracy of these approaches. As observed, the proposed LFP_H can be considered an ϵ-optimal solution, offering the best tradeoff between accuracy and computational time. Indeed, in scenario where we cannot afford the computational cost of solutions like LFP_CG, which has an average accuracy of approximately 11%, LFP_H provides a still-strong average accuracy of around 14% with significantly faster running times. The proposed LFP_H also proves to be an optimal choice during low traffic periods, such as around 6AM in our input data, where it offers both faster performance and higher accuracy.

4.5.5 Path Delay of LFP_CG vs. LFP_H

Figure 4.12 illustrates the delay requirements as well as the path delay distribution for requests from each service class over a 24-hour period.

As shown, both LFP_CG and LFP_H are able to find solutions that ensure the overall sparse delay exceeds 50% of the required delay for all traffic types. This is particularly advantageous, as it guarantees that the delay Quality of Service (QoS) is consistently met, a critical factor in ensuring optimal user experience and maintaining service reliability across varying traffic conditions. Maintaining a delay within the required thresholds is especially important in applications where real-time performance is essential, such as video streaming or online gaming. Furthermore, the path delay for LFP_H is closely aligned with that of LFP_CG,

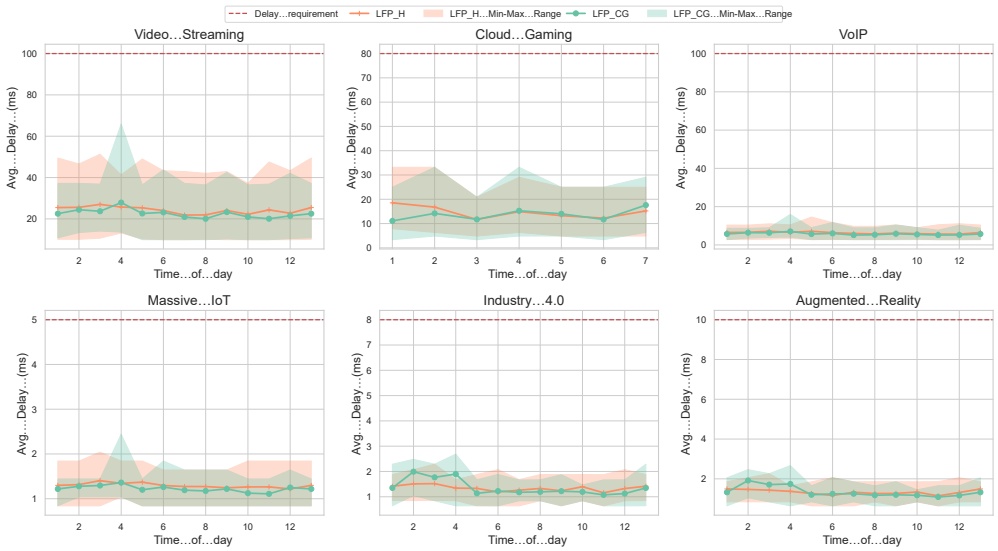
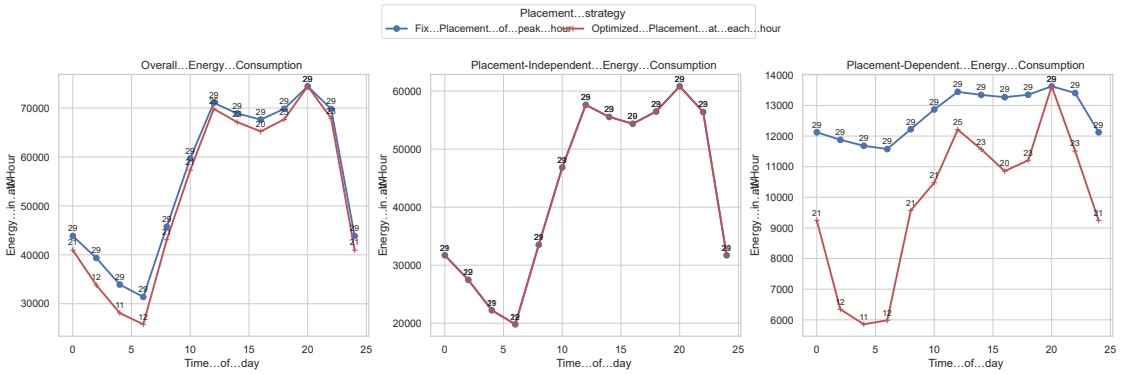


Figure 4.12: Path delay distribution from Source to destination: A comparison of proposed methods (CG, heuristics) and baselines.

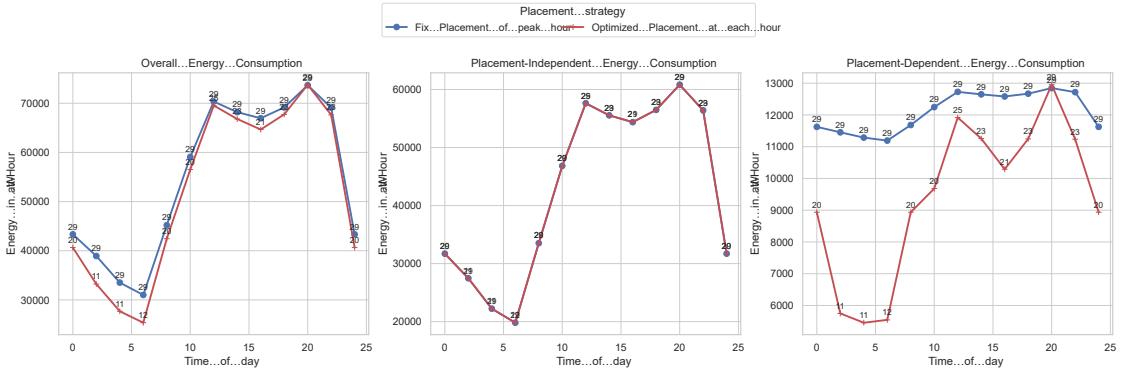
underscoring LFP_H's ability to deliver competitive results while being computationally more efficient. This highlights the potential of LFP_H to be deployed in environments with limited computational resources or real-time processing constraints. The close performance of LFP_H to LFP_CG suggests that, while LFP_CG may provide better results in certain scenarios, LFP_H offers a more practical alternative for large-scale deployments where computational overhead and time efficiency are crucial.

4.5.6 Fixed Placement based on Peak Time vs. Dynamic Placement Optimized for each Time Period

In this section, we compare the energy consumption of two placement strategies over a 24-hour period: fixed placement and dynamic placement. The fixed placement strategy assumes that the network functions remain in predefined locations throughout the day, while dynamic placement adjusts the allocation of resources in response to variations in traffic patterns. Typically, the operator configures the network to handle peak demand by optimizing the placement of resources for the most demanding time periods. This optimized placement is then applied throughout the day. We evaluate the performance of this static configuration against the dynamic approach. As expected, Figure 4.13 shows that with both LFP_CG and LFP_H, dynamic placement, by adapting to fluctuating demand, proves more effective in optimizing energy usage compared to fixed placement, which may lead to inefficiencies due to its static nature. However, in certain



(a) Energy consumption of LFP_H.



(b) Energy consumption of LFP_CG.

Figure 4.13: Comparison of energy consumption: fixed placement based on peak time vs. dynamic placement optimized for each time period.

scenarios, network operators may choose a static placement when the cost or complexity of implementing dynamic placement outweighs the benefits, or when specific operational constraints are present. These results highlight the importance of selecting the most efficient static placement to minimize energy consumption and reduce the performance gap relative to dynamic placement. This can ultimately lead to more sustainable and cost-effective network operations, especially in environments where dynamic placement may not be feasible.

4.5.7 Is the Static Placement based on Peak time Always the Best?

Building on the previous discussion of static and dynamic placement strategies, we now investigate whether static placement based on peak time is always the most efficient choice. Network operators typically optimize placement for peak demand periods, but we aim to assess whether this fixed placement remains the best choice over a 24-hour period. To explore this, we consider a placement strategy based on the

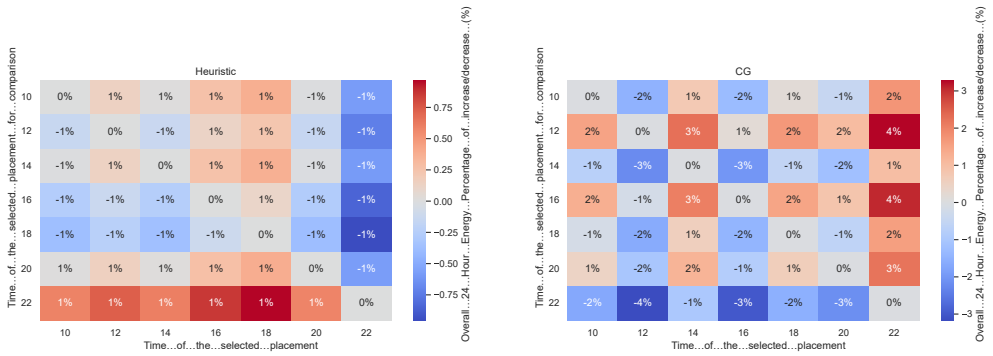


Figure 4.14: Percentage change in energy consumption relative to different fixed placements: The placement at time x on the x-axis is fixed, and compared to the placement at time y on the y-axis. The percentage indicates the increase or decrease in energy consumption relative to the x-axis placement. For example, at peak time $x = 20$, using the placement at time $y = 22$ results in a 1% increase in total energy over a 24-hour period with the heuristic method, while it leads to a 3% decrease in energy with the column generation method. This suggests that the peak-time placement is not always optimal, as it can lead to higher overall energy consumption compared to placements at other times.

optimal configuration for each hour of the day and use it as a static placement across the entire 24-hour time window. If this placement provides a feasible solution across all traffic conditions within the 24-hour period, we compute the overall energy consumption for that time frame. We then compare the energy consumption of this static placement against the results from other feasible placements within the same 24-hour period. Figure 4.14 presents the percentage change in energy relative to different fixed placements for pairs of hours. A positive percentage indicates an increase in energy consumption, while a negative percentage signifies a reduction in energy. This analysis helps determine whether the traditional approach of using a fixed placement based solely on peak time truly leads to the most energy-efficient solution or if there are other placements that offer better performance throughout the day. As shown in Figure 4.14, both the solutions derived from LFP_H and LFP_CG demonstrate that the placement optimized for peak demand time is not necessarily the best choice for overall performance across the entire 24-hour period. Specifically, the placement found by LFP_H for the traffic at hour 6pm offers the most energy-efficient solution when applied throughout the day. Using this placement results in a decrease of 1% in energy consumption when compared to placements based on 8pm ($x = 20$) peak time and other times with feasible solution i.e., 10am ($x = 10$), 12pm ($x = 12$), 2pm ($x = 14$), 4pm ($x = 16$), and 10pm ($x = 22$). The placement at 6pm ($x = 18$) shows a better balance of energy efficiency across varying traffic patterns. Similarly, for the solution provided by LFP_CG , using the placement at 10pm ($x = 22$) yields the most significant benefits. In this case, the placement based on peak time 8pm results in a 3% increase in energy consumption, while

using the placements from hours 10am ($x = 10$), 12pm ($x = 12$), 2pm ($x = 14$), 4pm ($x = 16$), and 6pm ($x = 18$), results in increases of 2%, 4%, 1%, 4%, and 2%, respectively. These results suggest that while peak-time placements are optimized for specific high-demand periods, they do not offer the most energy-efficient performance when applied to the entire day.

These findings underscore the importance of considering more than just peak traffic periods when determining the optimal placement of network functions. Both LFP_H and LFP_CG highlight that placements from non-peak periods can often result in more favorable energy consumption outcomes over the course of a day. This insight is valuable for network operators who aim to balance energy efficiency with operational flexibility, suggesting that dynamic or non-peak-based static placements could be more effective than traditional peak-based configurations. Further, these results emphasize the need for advanced optimization strategies that take into account traffic variability over time, thereby ensuring more efficient resource utilization and energy management throughout the network's operational period.

4.6 Conclusion

In this study, we modeled the placement of DU, CU, and UPF as a large-scale integer linear program, solved via decomposition (LFP_CG) and heuristic (LFP_H) methods. Our solutions jointly optimize function placement and routing, ensuring low-latency provisioning and high QoS, as demonstrated using real traffic data from Montreal. We achieved up to 14% energy savings over the baseline. While LFP_CG yields optimized results, LFP_H offers a scalable and efficient alternative for large deployments. Our findings also show that static, peak-time-based placements can lead to inefficiencies across the day, highlighting the need to consider broader temporal patterns. This work provides practical insights for balancing static and dynamic strategies in network optimization.

Chapter 5

An Asynchronous scheme for Convergence time and Energy Management in Distributed ML

This chapter presents the proposed solution aimed at addressing key challenges in Distributed Machine Learning (DML), including slow convergence, high energy consumption, and device/data heterogeneity. It introduces AFSL, a fully asynchronous federated split learning framework that combines the strengths of federated learning and split learning to accelerate convergence in heterogeneous environments. To further enhance energy efficiency, an improved variant called AFSL+ is proposed, incorporating additional mechanisms to reduce computational and communication costs.

5.1 Introduction

The rapid evolution of wireless communication technologies has paved the way for a new era with the advent of fifth-generation (5G) networks and the emerging sixth-generation (6G) paradigm. These next-generation networks promise unprecedented capabilities such as ultra-low latency, massive connectivity, high data rates, and seamless integration of intelligent services. These features enable diverse applications including massive Internet of Things (IoT) deployments, network function virtualization (NFV), and real-time network optimization driven by machine learning (ML) techniques [Li et al., 2018b]. Artificial Intelligence (AI) and ML are becoming foundational tools for network automation, performance enhancement, and

improving user experience in 5G and beyond [Stuart et al., 2023]. In current 5G deployments, AI integration typically takes three forms: (i) replacing traditional rule-based algorithms, (ii) introducing new AI-driven components, and (iii) adding intelligent control mechanisms to legacy systems. Looking forward, 6G networks are envisioned as *AI-native*, where AI is deeply embedded into the core of network functions and control mechanisms. However, the AI-driven evolution of networks introduces new challenges. These include managing *heterogeneous devices* with diverse computational capabilities, and processing *distributed, large-scale, and heterogeneous data* generated at the network edge. Traditional ML training paradigms, which depend on centralized data aggregation and cloud-based training, are increasingly inadequate. Centralized training not only incurs high communication latency but also raises critical concerns around privacy and energy efficiency, especially as training data continues to grow in size and sensitivity [Duan et al., 2022].

To overcome these limitations, *Distributed Machine Learning (DML)* approaches have emerged as promising alternatives. Techniques such as *Federated Learning (FL)* [Konečnỳ et al., 2016] and *Split Learning (SL)* [Gupta and Raskar, 2018] have gained traction by enabling model training across multiple devices without centralizing data. FL allows devices to train models locally and aggregate updates at a central server, while SL partitions the model between clients and the server, reducing computational burdens on resource-constrained devices.

Despite their advantages, FL and SL face critical challenges in real-world deployments. These include: Device and data *heterogeneity*, which slows down convergence and reduces model generalizability, *Energy constraints*, particularly on mobile and edge devices, and High *network overhead* due to frequent communication rounds. Existing efforts to improve energy efficiency in FL [Arouj and Abdelmoniem, 2024] and SL [Li et al., 2024a] predominantly rely on *synchronous* training schemes. However, synchronous protocols are vulnerable to delays caused by *straggler devices*, leading to inefficient use of network and computational resources. The recent trend of combining FL and SL into hybrid frameworks [Turina et al., 2021; Thapa et al., 2022; Jeon and Kim, 2020] has highlighted a critical gap: *energy-efficient asynchronous* learning protocols remain largely unexplored.

This chapter addresses that gap by proposing a novel framework called **AFSL** (Asynchronous Federated Split Learning). AFSL is designed to accelerate convergence while maintaining model accuracy and reducing energy consumption in heterogeneous and distributed environments. It does so by allowing clients to train their partial models asynchronously, thus avoiding the bottlenecks caused by slower devices, while

leveraging the structural benefits of both FL and SL paradigms. To further enhance energy efficiency and training performance, we introduce an enhanced variant called **AFSL+**. This extension incorporates a *client selection strategy* that operates in two stages: Devices are clustered based on *label distribution* and *data entropy*, ensuring that each group contains devices with similar data profiles. Within each cluster, a representative client is selected based on its *energy profile*, enabling training on a small but representative subset of the data while minimizing energy usage. By combining intelligent device selection with asynchronous parallelism, AFSL+ achieves substantial reductions in communication overhead and energy consumption, while accelerating model convergence. The *novelty* of AFSL+ lies in its ability to strike a balance between **energy efficiency and model performance** in asynchronous, hybrid FL-SL systems, a problem not adequately addressed in current literature. Through adaptive client clustering and energy-aware selection, AFSL+ ensures efficient distributed training without compromising the accuracy or robustness of the global model.

5.2 Related Work

This section reviews the state-of-the-art in hybrid federated and split learning frameworks and energy-efficient strategies in distributed machine learning.

5.2.1 Hybrid Federated and Split Learning Approaches

In the literature, two primary paradigms for distributed machine learning (DML) have emerged: *Federated Learning* (FL) and *Split Learning* (SL). Federated learning [Konečnỳ et al., 2016] enables multiple clients to collaboratively train a global model using their local datasets. Each client trains a model locally, and the resulting updates are aggregated by a central server. While FL is scalable and preserves data privacy to some extent, it still faces challenges related to computational burden on edge devices [Duan et al., 2022] and potential privacy risks due to gradient leakage [Turina et al., 2021]. To alleviate these issues, split learning [Gupta and Raskar, 2018] was proposed. SL divides a deep neural network into client-side and server-side components, enabling computational offloading to the server while keeping raw data local. This reduces the burden on clients and enhances privacy [Duan et al., 2022]. However, SL is typically implemented in a sequential manner, which can lead to under-utilization of resources and scalability limitations. Moreover, SL's performance may degrade in the presence of non-independent and identically distributed

(non-IID) data [Turina et al., 2021]. To leverage the strengths of both FL (scalability) and SL (reduced client computation and stronger privacy guarantees), hybrid FL-SL approaches have recently gained attention. Parallel Split Learning (PSL) [Jeon and Kim, 2020], Split Federated Learning (SFL) [Thapa et al., 2022], and its generalization Split Federated Learning Generalization (SFLG) [Gao et al., 2021] are notable examples. These frameworks aim to combine FL's aggregation mechanism with SL's split model architecture. In [Turina et al., 2021], authors proposed the Federated Split Learning (FSL) framework, where each client maintains its own server-side model replica. Unlike SFL and SFLG, FSL only synchronizes the server-side model and does not share client-side weights, thereby enhancing privacy. However, the synchronous aggregation of server-side models at the end of each epoch can lead to inefficiencies in heterogeneous environments. Devices with greater computational or networking capabilities may remain idle, waiting for slower clients to finish, thus reducing overall system efficiency. Despite ongoing efforts to improve hybrid FL-SL systems, several challenges remain unresolved, particularly in scenarios involving heterogeneous devices and non-IID data distributions, such as those found in 5G and beyond networks with diverse IoT clients. While some recent studies have explored asynchronous schemes in FL [Xu et al., 2023] and SL [Zhou et al., 2022] independently, very few have attempted to integrate both learning paradigms in a fully asynchronous manner. Existing methods that do attempt such integration often suffer from suboptimal resource utilization and prolonged training times.

5.2.2 Energy-Efficient Distributed Learning

Energy efficiency has been the focus of several studies in federated and split learning, yet these works predominantly assume a synchronous training paradigm. The majority of energy-related research in FL targets energy-constrained mobile devices (e.g., battery-powered clients). In this context, strategies have been developed to accommodate limited training capacity or battery constraints. For instance, [Zhao et al., 2022] proposed *FedNorm*, which selects energy-sufficient clients based on local model weight updates. [Maciel et al., 2024] introduced the *Energy Saving Client Selection (ESCS)* method, prioritizing clients first based on their energy availability, then by their contribution to model performance. [Savoia et al., 2024] developed a selection process that considers both the entropy of client-held data and their energy reserves. These methods all report trade-offs between accuracy and energy savings, with some cases showing convergence challenges under severe energy constraints. A second line of research aims to minimize overall energy consumption using various strategies, again under synchronous setups. Key methods include optimizing the

selection of the cut layer in SFL to reduce client energy consumption [Lee et al., 2024], enabling data sharing among clients (with potential privacy compromises) [Shullary et al., 2022], and exploring edge-cloud offloading strategies [Arouj and Abdelmoniem, 2024]. Additionally, [Li et al., 2023] proposed a client selection mechanism based on gradient descent rate. Some studies also address energy optimization from a system-wide perspective, focusing not on device-level constraints but on optimizing convergence time and training rounds. For example, [Zhao et al., 2022] proposed selecting only a subset of clients with significant model updates at periodic intervals to reduce energy usage, though their analysis is qualitative. [Luo et al., 2022] and [Lai et al., 2021] focused on client sampling strategies to reduce wall-clock time, without directly quantifying energy savings. [Li et al., 2022a] proposed *PyramidFL*, which performs utility-based client selection from the server's perspective, followed by local optimization. Though effective in improving convergence, its energy implications remain unexplored. A comprehensive review [Fu et al., 2023] categorizes various client selection algorithms designed to address heterogeneity and improve FL performance. However, energy savings remain an underexplored metric in these approaches.

Summary and Research Gaps

This section reviewed the state-of-the-art in hybrid federated and split learning frameworks and energy-efficient strategies in distributed machine learning. While existing works have made significant strides, they fall short in several key areas:

- **Lack of Asynchronous Hybrid FL-SL Frameworks:** Current hybrid models rely predominantly on synchronous aggregation, which limits scalability and efficiency in heterogeneous environments.
- **Limited Focus on Energy Optimization Beyond Constraints:** Most energy-aware FL/SL studies aim to operate under energy limitations rather than proactively optimizing energy usage for performance improvement.
- **Poor Handling of Data Heterogeneity:** Existing methods inadequately address non-IID data distributions, which are prevalent in real-world scenarios such as IoT and mobile networks.
- **Ineffective Client Selection:** Client selection strategies rarely consider the joint impact of data distribution, entropy, and energy profile in a unified, intelligent manner.

Our Contribution: To bridge these gaps, we propose a novel asynchronous framework that combines FL and SL in a parallel training protocol (AFSL). We further enhance this framework with AFSL+, a client selection mechanism based on entropy and label distribution-aware clustering. Unlike prior methods that confine client selection within homogeneous clusters, our approach dynamically selects clients across clusters in each round. This mixed selection enhances the diversity and representativeness of each training round, reducing abrupt shifts in model behavior and improving training stability. Additionally, by promoting data heterogeneity within each round, our framework mitigates excessive policy oscillations, an issue common in existing cluster-based selection schemes. As a result, our approach not only reduces energy consumption and communication overhead, but also accelerates convergence and enhances global model accuracy in heterogeneous, privacy-sensitive environments, offering a comprehensive solution for next-generation distributed learning systems.

5.3 AFSL: Enhancing Federated Split Learning with an Asynchronous Scheme

This section outlines the proposed asynchronous federated split learning scheme, detailing the system model, asynchronous update principle, and training procedure.

5.3.1 System Model and Proposed Asynchronous Scheme

System Model

In a given communication network, we consider a scenario in which N heterogeneous devices collaborate with an edge server to train a deep neural network model in a distributed manner. Each device i has its own local data set D_i, and the data sets are distributed in a non-IID fashion across devices. As in a real-world scenario, devices may have relatively limited and different computational capabilities, with some devices being more powerful than others. The architecture of the deep neural network is composed of several layers which are partitioned in two parts. As shown in Figure 5.1, each device executes the first partition, usually composed of a few layers with weights W_{c_i}. The second partition, composed of more layers with weights W_s, runs on a remote server (for example at the edge of the network). At the server side, each device i is trained with a server-side copy W_{s_i} of the global server weights W_s. At the end of each epoch, the trained copy W_{s_i} is used to update the global server weights W_s in an asynchronous way, as described in the next section.

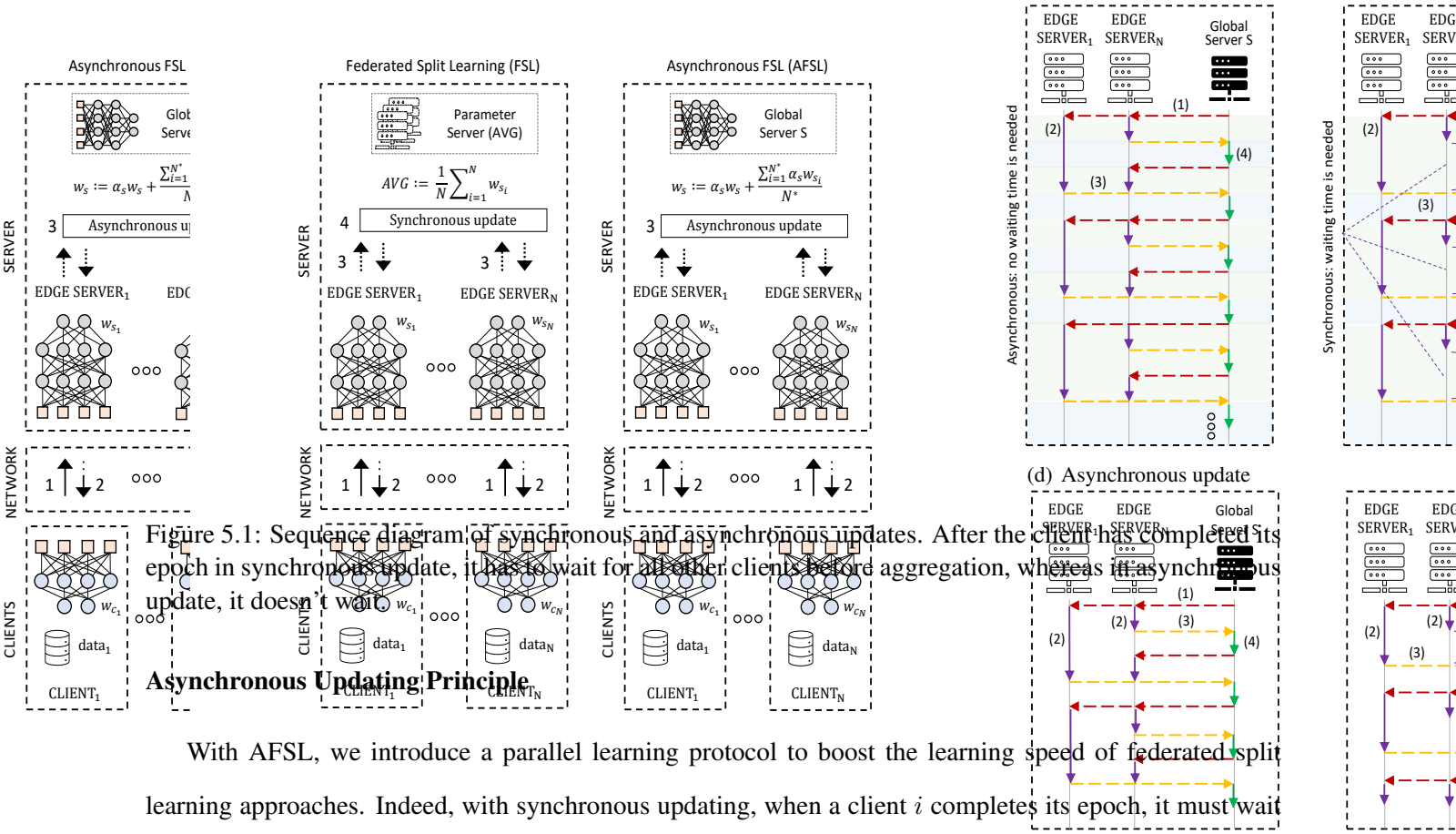


Figure 5.1: Sequence diagram of synchronous and asynchronous updates. After the client has completed its epoch in synchronous update, it has to wait for all other clients before aggregation, whereas in asynchronous update, it doesn't wait.

Asynchronous Updating Principle

With AFSL, we introduce a parallel learning protocol to boost the learning speed of federated split learning approaches. Indeed, with synchronous updating, when a client i completes its epoch, it must wait until all other clients participating in the training to complete their corresponding epoch before it can receive the aggregated weights W_s and move on to the next epoch. Many real-world environments are dynamic and heterogeneous. Devices may have different computational capabilities and network-varying conditions (e.g., latency, bit rate). In such a scenario, the slowest device, i.e. the one that ends its epoch last, will determine the training duration until convergence. This will lead to resource under-utilization and longer training times. This problem can be alleviated by introducing asynchronous updating. Figure 5.1(b) illustrates the AFSL architecture and its operation. Similary to FSL, each client has one exclusive edge server and exchanges intermediate data and gradients over the network (1,2). However, contrary to FSL, in AFSL the edge servers' weights are updated independently, in an asynchronous way (3). In other words, after the last batch of data (end of an epoch) of a given client, the client may be ready to perform aggregation, depending on the adopted strategy. To better understand this new mechanism, Figure 5.1(d) shows a sequence diagram of the asynchronous update. After receiving the global server weights *(1)*, the client completes its epoch *(2)*, and can directly submit its trained weights *(3)* to aggregation *(4)*. Then, the client moves on to the next epoch without being affected (put into inactive mode) by the training speed of other devices, as is the case of

synchronous updating shown in Figure 5.1(c). As a result, in asynchronous update, faster clients terminate their total epoch earlier, thus consuming fewer resources.

Remark 1. Asynchronous aggregation principles

It is important to note that in AFSL, asynchronous aggregation can be performed using various strategies tailored to specific use cases or desired design preferences. Therefore, asynchronous updates can be implemented based on different principles, which makes them highly adaptable to various device heterogeneity situations and advantageous in various scenarios. Model performance can then be further improved using different aggregation principles, including:

- directly when a model ends its epoch or,
- after receiving a given percentage of models,
- after a certain time threshold.

Therefore, at any point during the training process, the global server weights W_s can be updated with the number of models ready for aggregation ($N^\star$), utilizing an aggregation equation such as:

$$W_s \leftarrow \alpha_s W_s + \frac{\sum_{i=1}^{N^\star} \alpha_{s_i} W_{s_i}}{N^\star}, \tag{5.1}$$

where $N^\star \leq N$ is the number of clients ready for aggregation at a given time based on the chosen asynchronous aggregation principle. α_s and α_{s_i} are, respectively, the component-specific weights regarding the global server and the edge server of each client i. These two parameters are used to control the update weighting. They are defined so that $\alpha_s + \sum_{i=1}^{N^\star} \alpha_{s_i} = 1$, i.e., these parameters are not set as a unique constant but are varying based on N^* (e.g., considering a case where α_s and all the α_{s_i} are the same, we have $\alpha_s = \alpha_{s_i} = \frac{1}{N^*+1}$).

Remark 2. Dynamic update of α_s and α_{s_i} parameters

As previously mentioned, the number of models ready for aggregation N^* is directly influenced by the chosen aggregation principle, leading to different ranges of values for N^*. This introduces instability in weight aggregation if the weighting parameters, α_s and α_{s_i}, are not carefully selected. For instance, when opting for the design choice where $\alpha_s = \alpha_{s_i} = \frac{1}{N^*+1}$, i.e., assigning equal attention to both parameter server weights and client-ready model weights, there is a risk of instability, particularly when W_{s_i} distributions significantly differs from W_s distribution and N^* is small (e.g., $N^* \leq 2$). Given the potential for instability,

ensuring a stable model and preventing abrupt changes in the parameter server W_s, which could adversely impact the training of other clients, underscores the critical importance of carefully choosing the parameters α and α_{s_i}.

It is noticeable that a great number of combinations can be done with α_s and α_{s_i} parameters. To attain performance comparable to synchronous approaches (e.g., FSL) while minimizing convergence time, a key attribute of our proposed AFSL lies in how we aggregate the ready models with the global parameter server. Indeed, we proposed implementing a dynamic parameter updates, wherein α_s and α_{s_i} can vary based on the state (i.e., maturity) of the parameter server model at each aggregation step. This variation could be tied to factors such as the client epoch or the server aggregation number. We proposed a formulation where the edge servers' weight α_{s_i} exponentially decays over the number of server aggregations referred to as e_s. In other words, depending on the server's maturity (the higher the aggregation step, the more mature the global server model is), more weight is given to the global server during the aggregation. We propose to define the edge server weights α_{s_i} and the global server weight α_s as follows:

$$\alpha_{s_i} = \frac{\sigma\, \beta^{e_s}}{N^\star}, \qquad\qquad \alpha_s = 1 - \sum_{i=1}^{N^*} \alpha_{s_i}, \tag{5.2}$$

where σ is a parameter that can be manually updated to control the effect of the decay β^{e_s}, with $\beta < 1$ being the decay rate. The parameters σ and β were chosen with a grid search and the selected values were $\sigma = 0.5$ and $\beta = 0.99$.

AFSL Training Procedure

As presented in Algorithm 5.1, AFSL operates as follows. First, the main process initializes the global server weights W_s according to a weight initialization strategy. Then, in parallel, the training of each connected client i with a remote server copy W_{s_i} of W_s is established whereby the weights of the client's local model W_{c_i} are initialized according to the weight initialization strategy. Next, the local client model i performs forward propagation and sends its output c_output_i, i.e., the smashed data, and labels $labels_i$ to the server. The server terminates the forward propagation with the corresponding server copy W_{s_i} and gets the output $output_i$. The server applies a loss function, e.g., sum of squares or cross entropy, to the output $output_i$ to accumulate the errors $Loss_i$. The gradient $grad_i$ of the error is back propagated through the

Algorithm 5.1 Asynchronous Federated Split Learning (AFSL)

procedure CLIENT ▷ running at client side
  Initialize client i weights W_{c_i} according to a weight
  initialization strategy.
  for $epoch$ in $epochs_i$ **do**
    for $data_i, labels_i$ in dataset D_i **do**
      $c\_output_i \leftarrow c\_Forward_i(data_i)$
      send $(c\_output_i, labels_i)$ to server
      $s\_grad_i \leftarrow$ receive gradient form server
      $c\_Backprop_i(s\_grad_i)$
    end for
  end for
end procedure
procedure SERVER ▷ running at server side
  for $epoch$ in $epochs_i$ **do**
    $W_{s_i} \leftarrow$ get latest server weights W_s
    for $batch$ in num_batch_i **do**
      $c\_output_i, labels_i \leftarrow$ receive output and labels
      from client i
      $output_i \leftarrow s\_Forward_i(c\_output_i)$
      $Loss_i \leftarrow LossFunc(output_i, labels_i)$
      $grad_i \leftarrow Gradient(Loss_i)$
      $s\_grad_i \leftarrow s\_Backprop_i(grad_i)$
      send s_grad_i to client i
    end for
    submit weight W_{s_i} for aggregation to main process
  end for
end procedure
procedure MAIN ▷ running at server side
  Initialize global server weight W_s according to a
  weight initialization strategy.
  for each connected client i, **in parallel do**
    Establish training with server by executing
    procedures `CLIENT()` and `SERVER()`
  end for
  while Training ongoing **do**
    get current submitted weights from clients
    $W_s \leftarrow$ update server weight (according to Section 5.3.1)
  end while
end procedure

server layers, and the server output gradient s_grad_i is sent to the client, which terminates its back propagation. Once a client completes an epoch, as opposed to a synchronous update involving the aggregation of all models, its trained copy server weights W_{s_i} are used to update the global server model weights W_s according to an asynchronous update principle as defined in Subsection 5.3.1. Thus, leading to accelerated training. Equation (5.1) shows the general asynchronous aggregation scheme for the global server model, where the parameters α_s and α_{s_i} are dynamically chosen to enhance model stability and performance. The next epoch continues with W_{s_i} updated to the latest W_s, and this is done in parallel for all clients.

5.3.2 Performance Evaluation

To quantify the effectiveness of the proposed AFSL, we conducted three experiments, namely Experiments I to III, under different conditions to evaluate training time, model accuracy, and robustness under a

dynamic environment. We assume a scenario where a group of N clients train, in a cooperative way, models for a classification and regression problem while keeping their data locally. This context may well suit a wide range of IoT applications, such as home automation, smart security, smart vehicles, speech commands processing, etc. To reproduce such a scenario, for the classification problem, we adopted the well-known MNIST dataset [LeCun and C, 2010] with the LeNet-5 Neural Network [LeCun et al., 1998] and we considered accuracy as the performance metric for this dataset. On the other hand, the evaluation of the regression problem is based on realistic video on demand (VoD) traces [Samani, 2023] obtained from a testbed at KTH University. Each trace group contains about 1700 features (e.g., CPU utilization per core, memory utilization, and disk I/O) collected from six server machines. Three VoD service-level metrics serve as the target values, i.e., net value of read average delay, average display delay, and average audio play delay. More details about these traces can be found in [Yanggratoke et al., 2018]. We used the normalized mean absolute error as the performance metric to evaluate the models trained with this dataset.

All experiments, were run on a single device with multiple CPUs, emulating a distributed system. The clients and the server were deployed using different terminals, and socket communication was used to guarantee intermediate data transmission between clients and server. All experiments were conducted on a AMD Ryzen Threadripper 2990WX-32-Core processor @ 2.95GHZ with 128 GB of RAM under the Python environment version 3.10.6, with Pytorch 2.0.1 library.

FSL vs AFSL with Similar Clients

For Experiment I, we aim to compare AFSL and FSL under a scenario where the clients have similar computing capabilities and the same number of data samples. To do so, we adopted the MNIST dataset with a balanced IID distribution (as illustrated in Figure 5.2(a) and for the VoD dataset, as the data were non-IID by nature, we ensured the same number of samples per client (as illustrated in Figure 5.2(d)). We also set, empirically, the number of epochs to 30 and conducted three different simulations considering 5, 20, and 40 connected clients on the MNIST dataset and 8, 16, and 24 clients on the VoD dataset. It is worth mentioning that, in this experiment, all the clients begin the training at the same time. Moreover, to capture the deviation among different runs, all experiments were conducted five times, and results were averaged.

The time performance for each simulation is summarized in Figure 5.2(b) and Figure 5.2(e) for MNIST and VoD datasets, respectively. For both datasets, we can observe that the average training time per epoch is similar (i.e., time difference less than one second) for FLS and AFSL, with the time difference increasing

slightly as as the number of customers increases. Indeed, on the MNIST dataset, for 5, 20 and 40 clients, we observe that the average time necessary for the FSL to complete an epoch is increased by 0.31 s, 0.41 s and 0.54 s, respectively, compared to AFSL. This can be justified by the fact that the adopted multiprocessing scheme adds, by nature, some heterogeneity to the clients in terms of CPU capability. Therefore, in a scenario where the devices are homogeneous and the quality of connectivity to the server is identical, the devices end their epoch at approximately the same time. AFSL and FSL achieve similar learning times, with AFSL being slightly better than FSL, as it is difficult to achieve perfect homogeneity.

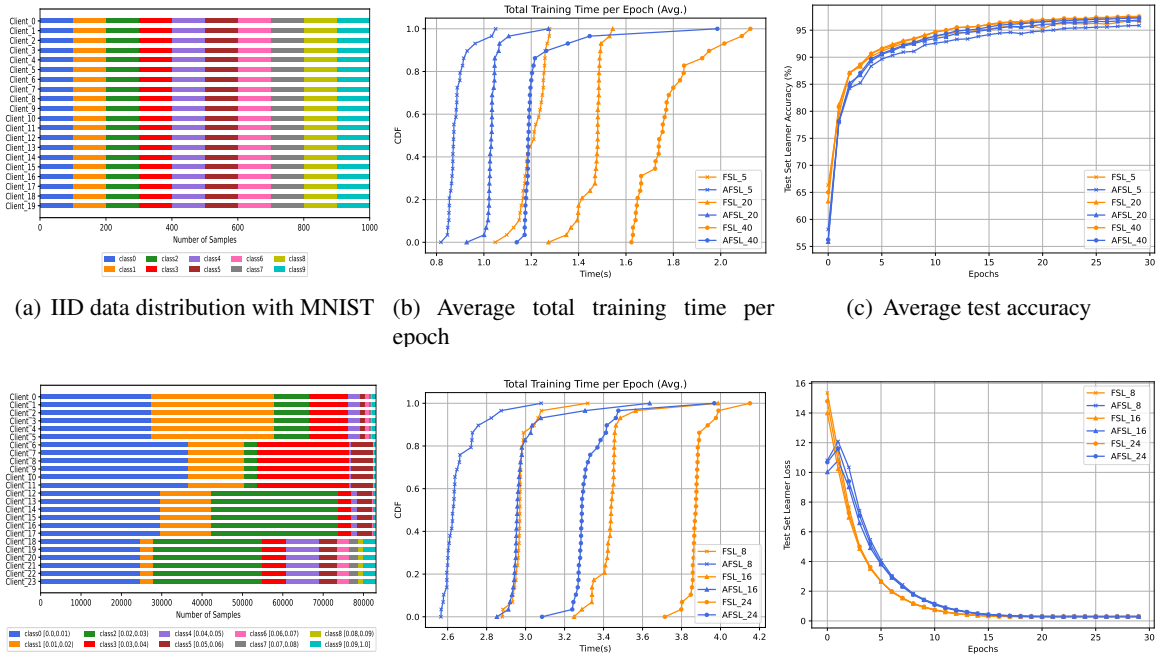
Regarding the accuracy of the models for the MNIST dataset and the loss (normalized mean absolute error) for the VoD dataset, respectively in Figure 5.2(c) and Figure 5.2(f), both approaches showed a similar learning curve for all simulations. However, on the MNIST dataset, a slightly worse performance at the end of the training was noted for AFSL when compared to FSL. The observed difference in terms of accuracy for 5, 20, and 40 clients is, respectively, $\approx$0.7%, $\approx$0.5% and $\approx$0.3%, which is quite negligible. Similarly, a negligible difference in loss is observed on the VoD dataset. This means that in this scenario, AFSL performs as effectively as FSL. The similarity in performance can be attributed to the aggregation scheme, which prioritizes a more stable update of the global parameter server, reflecting the characteristics of the synchronous scheme.

FSL vs AFSL with Heterogeneous Clients

The second experiment aims to analyze the performance of AFSL and FSL in a scenario with heterogeneous devices. In other words, devices may have different compute capabilities and the available number of data samples for each user may be different. This is a very common scenario in the real world, and it's essential that AFSL works well in this case.

To emulate such a scenario using the MNIST dataset, we adopted a non-IID data distribution whereby the number of data points and class proportions is unbalanced. Samples are partitioned per user i following a normal distribution $|N(\mu, \sigma^2)|$ per each class. The μ value was empirically set to 100 and σ to 0.5μ. To give a better intuition, Figure 5.3(a) illustrates the data distribution for a sample of 20 clients. As the VoD dataset is non-IID by nature, no modifications were required (as presented in Figure 5.3(d)).

To simulate different compute capabilities, we introduced some artificial delay in an experiment with 20 clients on MNIST dataset and 24 clients on VoD dataset. Although all clients simultaneously train their own model in a cooperative way, we equally divided them into four different groups. Each group had a



(a) IID data distribution with MNIST

(b) Average total training time per epoch

(c) Average test accuracy

(d) Modified VoD dataset with the same number of samples per client (values were divided into ten ranges)

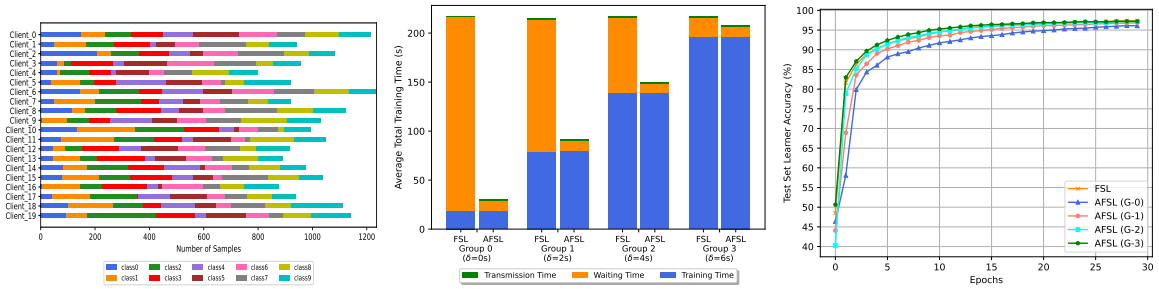
(e) Average total training time per epoch

(f) Average test loss

Figure 5.2: Experiment I – FSL & AFSL with 5, 20 and 40 clients on MNIST (Figures 5.2(a), 5.2(b) and 5.2(c)) and 8, 16, 24 clients on VoD (Figures 5.2(d), 5.2(e) and 5.2(f)). In all these experiments, clients have similar computational capabilities with the same number of data samples.

delay value based on a normal distribution $N(\mu_{delay}, \sigma^2_{delay})$. Therefore, devices in the same group emulate devices with similar compute capabilities, represented by the same μ_{delay}, and difference in capability within a group is expressed by the deviation σ_{delay}. Considering as a baseline $2\times$ the average training time of Experiment I ($\approx$1s for 20 clients on MNIST), and to emulate clients with considerable less computational resources, the μ_{delay} values were set to 0s, 2s, 4s, 6s, respectively for the four groups of clients. The σ_{delay} value was set to $0.05\mu_{delay}$, to limit the range of variability within a group. The first group had $\mu_{delay} = 0$s, therefore no extra delay was added for the first group.

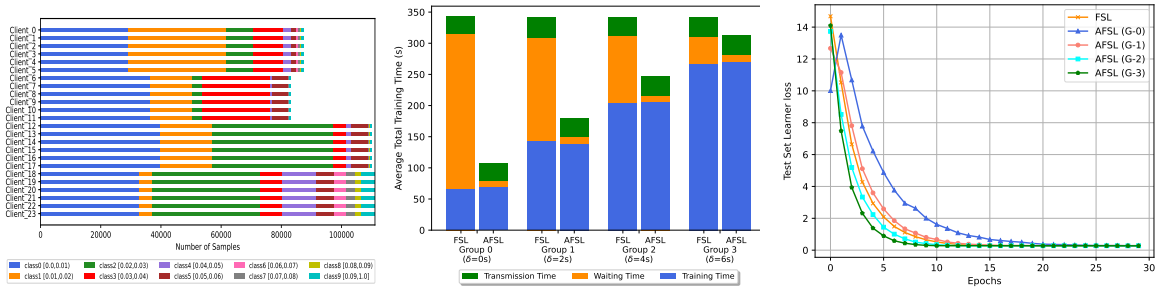
The results in terms of total training time per group are presented in Figure 5.3b and Figure 5.3e for MNIST and VoD dataset, respectively. When adopting AFSL on MNIST, a reduction in total training time of $\approx$86.5%, $\approx$57.5%, $\approx$31% and $\approx$4.6% was observed for groups 0, 1, 2 and 3, respectively. It can be observed that in a scenario with heterogeneous clients, the FSL approach has considerably poorer performance in terms of total time. Although the time spent on training (blue part) is similar in both approaches, in FSL the faster clients have to wait (orange part) for the slower ones to terminate their epoch in order to do the



(a) Non-IID data distribution with MNIST

(b) Average total training time

(c) Average test accuracy



(d) Original VoD dataset with different number of samples per client (values were divided into ten ranges)

(e) Average total training time

(f) Average test loss

Figure 5.3: Comparison between FSL & AFSL on MNIST (Figures 5.3(a), 5.3(b) and 5.3(c)) and on VoD (Figures 5.3(d), 5.3(e) and 5.3(f)) with heterogeneous clients. In this simulation, 20 clients were trained together but they were equally divided into four groups with different artificial delays. Non-iid data was adopted.

model aggregation and continue the training. On the other hand, in AFSL, each client can conclude the model aggregation step independently, right after they finish their epoch. Therefore, the average time to conclude the training is reduced, indicating that AFSL is better at exploiting all available resources. Similar results are obtained with the real-world VoD dataset, showing that the proposed solution is data agnostic. The communication time shown in green is much greater with the VoD dataset than with the MNIST dataset, due to the size of the smashed data being transferred in the network, which is higher with VoD than with MNIST.

In terms of accuracy on MNIST dataset, as presented in Figure 5.3(c), all groups reach 96% or more at the end of their training. In AFSL, groups 0, 1, 2 and 3, reach respectively, 96.05%, 96.84%, 97.1% and 97.2%, compared to 97.4% in the case of FSL. In AFSL, as the server model is updated as soon as the clients epochs are finished, the faster clients (i.e., clients of group 0, blue line) present the worst accuracy because they finish their epochs faster and, therefore, are exposed to less knowledge coming from the slower

clients (via global server S). On the other hand, slower clients present better accuracy because they take advantage of the multiple epochs performed by the faster clients. However, as the final model is defined based on the last version of the global server S, we can assume that AFSL showed a difference in accuracy of 0.2% when compared to FSL. On the VoD dataset, we observe a more stable loss at the end of training, which is similar for all the AFSL groups and with FSL. These results on the MNIST and VoD datasets demonstrate the effectiveness of AFSL, which brings a significant gain in terms of learning time compared with FSL, while maintaining good performance. The high-speed clients do not necessarily have to undergo additional training epochs. They can complete their training after completing the predetermined number of training epochs, ensuring that they obtain a good enough model. These clients will consequently utilize fewer resources, a vital characteristic for devices with limited computing capabilities, like IoT ones. It is noteworthy that with the asynchronous scheme, although it reduces the average training time, as well as the resource utilization, as observed, there may be a slight degradation in performance. Therefore, the selection of the aggregation principle helps mitigate the risk of accuracy degradation.

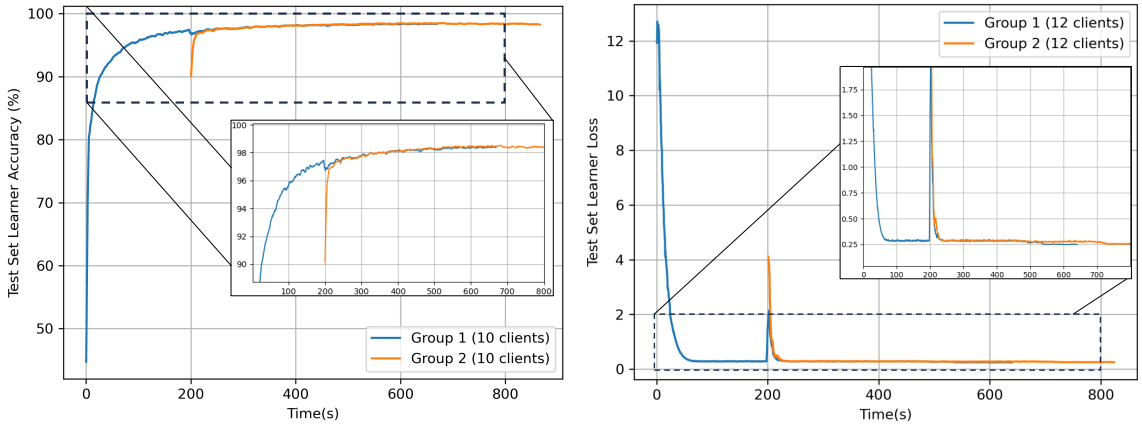
AFSL in a Dynamic Environment

In Experiment III, we study the performance of AFSL in a dynamic environment, where multiple clients with imbalanced data may connect/disconnect over time during the training phase. The dynamic nature is a common feature in various use cases for edge computing, spanning from smart factories and smart grids to autonomous vehicles and IoT applications. This experiment seeks to confirm the advantages of AFSL training in dynamic scenarios, i.e., provides robustness with a short recovery time by leveraging asynchronous client training. To replicate such a scenario, two sets of clients were simulated for each dataset. For MNIST, two groups of 10 clients each, and for VoD, two groups with 12 clients each. The first group begins training, while the second group joins the training after 200 seconds, once the learning curve of the first group stabilizes. The objective is to evaluate the impact of introducing new clients on the model's performance. To better observe the impact of the client dynamics, the number of epochs for each group was set to 130. To represent unbalanced data, the non-IID data distribution presented in Experiment II was adopted (see Figure 5.3(a) and Figure 5.3(d)). The learning accuracy results are presented in Figure 5.4. After joining the training, the second group takes advantage of the shared knowledge provided by the first group (via global server S) and learns faster.

For the MNIST dataset (Figure 5.4(a)), the first group starts the training with an average test accuracy of

44.5% which eventually stabilizes near to 97.44% after ≈200 seconds. On the other hand, the second group starts the training with 90.1% accuracy which eventually reaches the precision of the first group after ≈10 seconds. It is worth mentioning that when the second group connects, a reduction in accuracy of ≈0.7% is observed for the first group. This happens because the new group has not yet been through any back-propagation, and their gradients will behave more aggressively to reduce their loss, possibly affecting other clients. However, accuracy is recovered after a few seconds of training, when both groups begin to have similar average accuracy.

Similar results are observed on the VoD dataset, as shown in Figure 5.4(b), with the first group recovering its losses just seconds after the second group joined it. The proposed AFSL is therefore robust in a dynamic environment. Devices that join training late benefit from the knowledge acquired from other devices already participating in training, with a very short recovery time for the system as a whole This is a crucial attribute because in a distributed learning system, new, cutting-edge devices join ongoing training. It is therefore critical that their addition does not significantly degrade overall performance, with the need for a very short recovery time.



(a) MNIST dataset

(b) VoD dataset

Figure 5.4: Experiment III – AFSL performance analysis in a dynamic environment. In this experiment, two groups of clients starting the training in different times are trained together.

5.4 AFSL+: Enhancing Energy Management and Efficiency of AFSL

In this section, we present an enhanced framework of AFSL that integrates energy-awareness to further improve energy efficiency and management. We refer to this new design as AFSL+. It takes into account

the client energy profile, the performance progress (e.g., accuracy or loss), and data label distribution and entropy to optimize client selection in a distributed system.

5.4.1 AFSL+ Architecture

The high-level architecture of AFSL+ is shown in Figure 5.5, where we illustrate the original AFSL architecture along with an additional blocks designed to enhance energy efficiency. In AFSL, all devices train simultaneously, but this is not always required. By selecting a carefully chosen subset of clients in each round, similar performance can be achieved while minimizing energy consumption by reducing the time devices spend in active training mode. Thus, a trade-off between accuracy and energy efficiency must

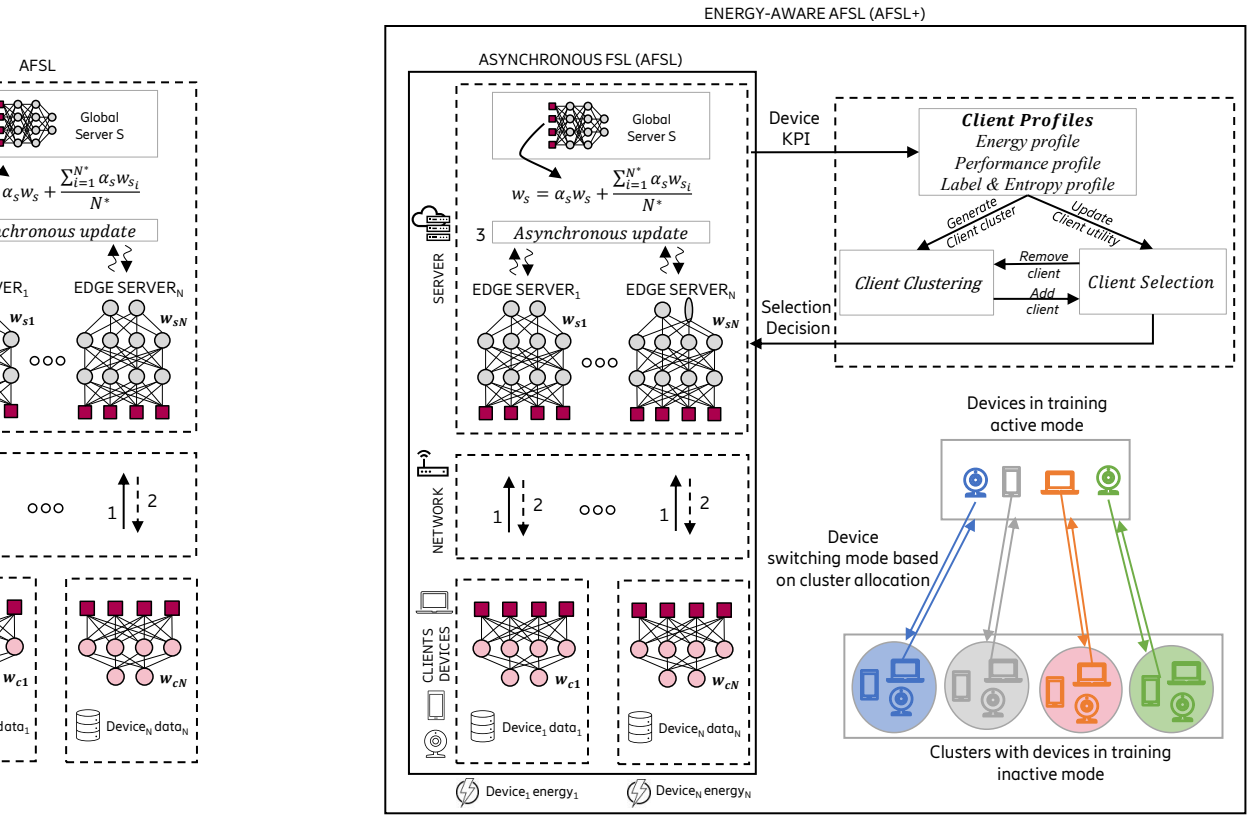


Figure 5.5: High-level architectural design of AFSL+: Device selection based on client profiles and their corresponding clusters.

With N devices participating to the training, AFSL+ addresses this by clustering the devices into K sets ($K < N$), and efficiently selecting clients from these clusters to participate in the training. Each cluster is formed based on similar label distributions and client entropy. One client is selected from each cluster to

form a set of K training active devices, while the remaining $N - K$ devices are in training inactive state. During training, in addition to transferring client activations to the server, each device n sends its energy consumption, denoted E_n, to the server at regular intervals, helping create an energy profile for each client. As training progresses asynchronously, once a device completes a set number of iterations, it is moved to the inactive set, and another client from the same cluster is selected to maintain K active device. This process continues until convergence or the desired performance is achieved.

Initialization

At the start of the training process, the server determines the number of iterations each client must complete before server global model averaging. Each client $n \in N$, hosting data D_n performs one iteration, after which the server collects the corresponding client labels $\{y_n^i\}_{i=1}^{|D_n|}$, the computed client loss L_n, and the received client energy E_n for each client. This information is then used to generate client profiles, which will guide the client selection process.

Clustering with Entropy and Label Feature

We perform clustering based on label distribution and client entropy, ensuring that each cluster contains clients with similar label and entropy distributions. The data D_n hosted by client n can be written as $D_n = \{D_{n,1}, D_{n,2}, \dots D_{n,|J|}\}$, where $D_{n,j}$ corresponds to the samples of class j, and $|J|$ represents the total number of classes across all clients. The features F_n of client n used for clustering can then be expressed as:

$$F_n = \left(H_n, \frac{|D_{n,1}|}{|D_n|}, \frac{|D_{n,2}|}{|D_n|}, \dots, \frac{|D_{n,|J|}|}{|D_n|} \right), \tag{5.3}$$

where $H_n = -\sum_{j=1}^{|J|} \frac{|D_{n,j}|}{|D_n|} \log(\frac{|D_{n,j}|}{|D_n|})$ is the entropy of client n.

For regression problems, where the labels are continuous values, we can discretize the label space into ranges or apply clustering techniques, such as DBSCAN, to determine the optimal number of clusters for the given data. This can then be used to construct the label distribution. The features of each client, as defined in (5.3), can subsequently be used to cluster clients with an unsupervised algorithm, such as KMeans.

Client Selection

After the first iteration, a set of K clients is selected. These clients are now active and participate in the training. To alleviate the label non-IID problem, we propose a sampling strategy in which one client is selected from each cluster. In fact, by sampling clients from different clusters, the data seen by the global model in each round will more effectively represent the union of all clients' data. This can then lead to faster convergence and improved performance. Once a client n has completed the allocated number of iterations, it is put into inactive mode and replaced by another client m from the same cluster as n.

In AFSL+, the algorithm identifies and prioritizes clients with minimal utility. Additionally, we employ an epsilon-greedy algorithm during client selection within each cluster to avoid local optima and facilitate exploration. The utility of each client consists of two components: one function measures system performance (accuracy or loss), and the other function quantifies the energy consumed by the client's device so far. Utility U_n for client n can be written as follows:

$$U_n = \alpha E_n + (1 - \alpha)1/L_n, \tag{5.4}$$

where $\alpha \in [0, 1]$ is the weight associated with the energy consumption of a client. E_n and L_n represent the normalized energy consumed by client n so far and its normalized training loss, respectively. We next detail the computation of these two Key Performance Indicators (KPIs).

Model Training

Each selected client performs forward and backward propagation, updating both the local model weights $\mathbf{W}_{c_n}$ and the server model weights $\mathbf{W}_{s_n}$. During forward propagation, activations are sent to the server, which computes the back-propagation gradients based on the received activations. For classification tasks, we use a calibrated loss function [Chen et al., 2022; Zhang et al., 2022] to enhance model performance under conditions of data heterogeneity. The calibration is performed using the logits computed by the server model. The loss function L_n for client n is defined as:

$$L_n = -\frac{1}{|D_n|} \sum_{i=1}^{|D_n|} \sum_{\ell=1}^{|J|} \mathbf{I}\{\ell = y_i\}. \log \left[\frac{e^{\bar{\ell}+\log P_n(\ell)}}{\sum_{j=1}^{|J|} e^{\bar{j}+\log P_n(j)}} \right], \tag{5.5}$$

where $\mathbf{I}(.)$ is the indicator function, $\bar{\ell}$ is the predicted score (i.e., output logit) for label ℓ, $P_n(\ell) = \frac{|D_{n,\ell}|}{|D_n|} + \delta$ approximates the class prior of label ℓ and $\delta > 0$ is a small constant added for numerical stability purpose.

In AFSL+, the energy consumption of each client is monitored in real-time and sent to the server at a specified frequency to update the client's utility. The energy E_n of client n that was inactive I times and active A times so far is estimated as:

$$E_n = \sum_{i=1}^{I} P^i_{n,idle} \times t^i_{n,idle} + \sum_{i=1}^{A} \sum_{c \in C} P^{i,c}_{n,busy} \times t^{i,c}_{n,busy}, \tag{5.6}$$

where for device n, $P^i_{n,idle}$ and $t^i_{n,idle}$ are respectively the power consumed and the time spent in the i-th idle state (i.e., training inactive). $P^{i,c}_{n,busy}$ and $t^{i,c}_{n,busy}$ refer to the power consumed and time spent in the i-th busy state (training active) of the c-th core in device n. Note that both $P^i_{n,idle}$ and $P^{i,c}_{n,busy}$ contains a computational power component and the communication power component of the device. If the execution system is CPU or GPU, $t^{i,c}_{n,busy}$ and $t^i_{n,idle}$ can be obtained from procfs[1] and sysfs[2] interfaces in the linux kernel. $P^{i,c}_{n,busy}$ and $P^i_{n,idle}$ for CPU/GPU are power measurements of the CPU/GPU at each frequency in the busy/idle states, respectively. Note that for battery-powered devices, measurement of the level of battery discharge can also be utilized.

5.4.2 Performance Evaluation

Datasets for Each Use Case

To evaluate the performance of the proposed AFSL+ approach, we consider two use cases. The first use case involves a non-IID MNIST [LeCun and C, 2010] dataset under label skewness for collaborative image classification. The second use case utilizes real-world video-on-demand (VoD) traces obtained from a testbed at KTH University [Samani, 2023], focusing on performance prediction in networking environments. Each VoD trace group includes approximately 1,700 features, such as CPU utilization per core, memory utilization, and disk I/O, collected from six server machines. The service-level metric for the VoD is the net value of the read average delay, which serves as our target value. Further details about these traces can be found in [Yanggratoke et al., 2018].

[1] https://man7.org/linux/man-pages/man5/proc.5.html
[2] https://man7.org/linux/man-pages/man5/sysfs.5.html

(a) Client server cluster connected to the edge server (b) Rack with the edge server

Figure 5.6: Overview of the testbed used for the evaluation

Experimental Setup

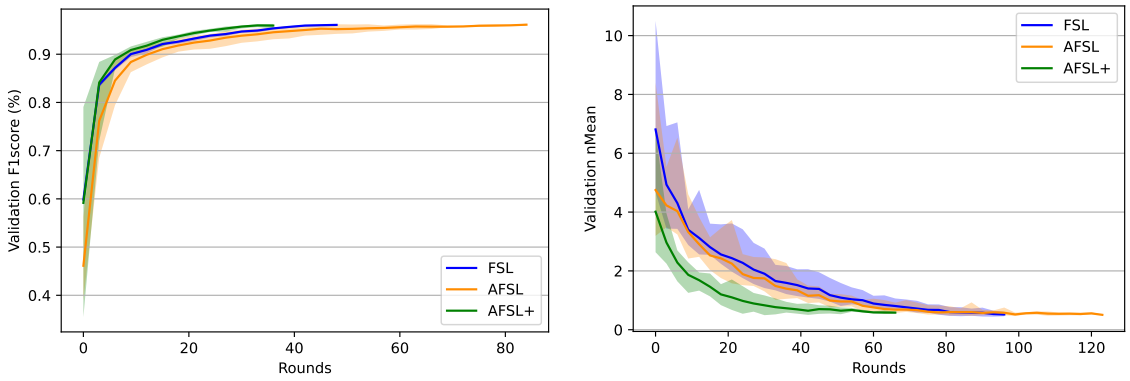
All experiments were carried out on a testbed comprising multiple client devices and a single edge server, as shown in Figure 5.6. Both the client devices (genuine lab PCs, AMD-based) and the edge server were configured with Python 3.7 and PyTorch 2.0. In each use case, a deep learning model was split into two parts: one deployed on the client side and the other on the server side. For the first use case, we employed the LeNet-5 neural network [LeCun et al., 1998] and for the second use case, we selected a 4 - layer DNN model, which was fine-tuned based on the dataset's characteristics. In both architectures, the model was cut after the first layer[3]. Performance for the first use case is evaluated using the F1-score, due to the dataset's imbalance, while the second use case uses normalized mean absolute error (nMeanAE) for regression evaluation. Performance is compared against two baselines: AFSL and FSL [Turina et al., 2021].

Training Convergence

We evaluate the training convergence of the proposed AFSL+ solution compared to the baselines. Figures 5.7(a) and 5.7(b) show the validation accuracy for the MNIST dataset and the validation nMean for

[3]Search for the optimal cut layer is outside the scope of this work.

the VoD dataset, respectively. It can be observed that in both use cases, AFSL+ reaches the target metrics (i.e., F1-score = 96% and nMean = 0.5) in fewer rounds (with asynchronous rounds counted each time the global server model is updated) compared to the two baselines. This improvement arises because AFSL+ intelligently selects clients for each training round, ensuring that the data distribution of active devices at any given time remains as representative as possible of the entire dataset across devices. By using only a carefully selected subset of clients each round, AFSL+ reduces noise in weight updates and improves model aggregation with more representative data. In contrast, AFSL requires more rounds than FSL due to its asynchronous scheme, which aggregates models as soon as a client finishes its local training, rather than waiting for all clients to complete training in the synchronous FSL scheme.



(a) MNIST Dataset

(b) VoD Dataset

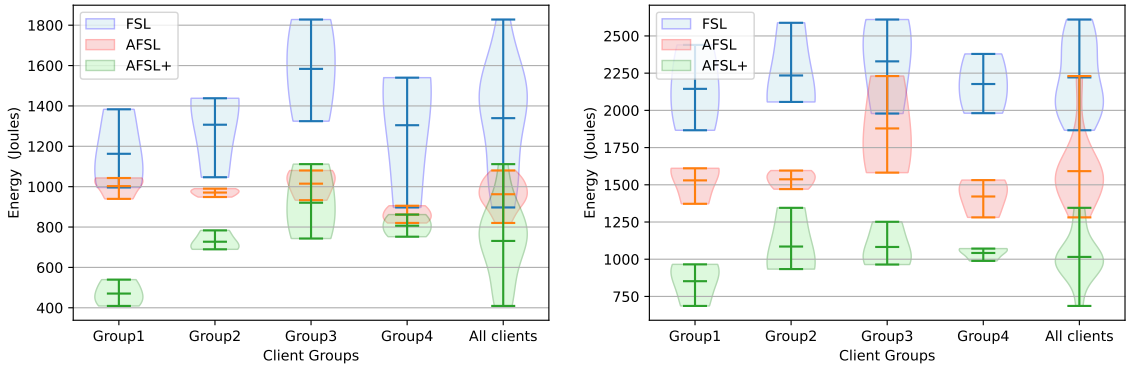
Figure 5.7: Validation accuracy for the MNIST dataset and normalized mean absolute error (nMean) for the VoD dataset, comparing FSL, AFSL, and AFSL+. The shaded region indicates the variation across devices.

Energy Analysis

The overall energy consumed by all the devices when running each method is presented in Figure 5.8. Devices are grouped into four categories based on their compute capabilities, reflecting the heterogeneity in training speeds, with some devices performing faster than others. As shown in both use case 1 (Figure 5.8(a)) and use case 2 (Figure 5.8(b)), AFSL+ consistently consumes less energy on average compared to AFSL and FSL. Specifically, we observe an energy reduction of $\approx 23\%$ compared to AFSL and $\approx 46\%$ compared to FSL for the MNIST dataset, and a reduction of $\approx 38\%$ compared to AFSL and $\approx 55\%$ compared to FSL for the VoD dataset.

This improvement is primarily due to two factors. First, unlike AFSL and FSL, AFSL+ does not keep

all clients active simultaneously; only a subset of clients participate in each round. Since inactive clients consume significantly less energy, this lowers overall consumption. Second, AFSL+ intelligently selects clients based on data distribution, performance, and energy usage, which can accelerate training and reduce the number of model updates needed for convergence, as shown in Figure 5.7. By using fewer rounds and alternating clients between active and inactive modes, AFSL+ significantly decreases overall energy consumption.



(a) MNIST Dataset

(b) VoD Dataset

Figure 5.8: Energy consumption of client aggregated based on their compute capabilities. Each violin plot indicates the distribution, the maximum, minimum and average value.

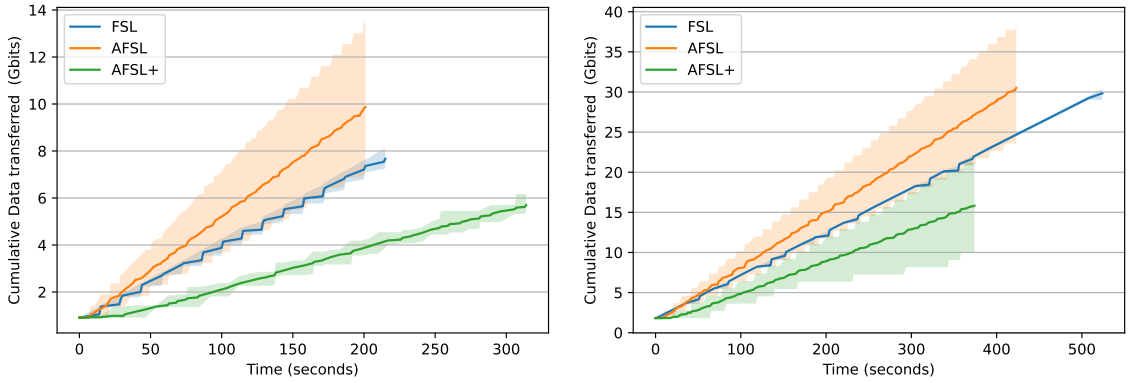
Communication Overhead vs. Training Time

Figure 5.9 shows the data transfer volume, which includes the communication cost in bits, accounting for the transmission of intermediate feature matrices, energy data from devices to the server, and gradient matrices from the server to the devices, until the target performance metric is achieved. The results demonstrate that AFSL+ transfers less data in both use cases. Specifically, on MNIST, AFSL+ achieves a reduction of $\approx 38\%$ and $\approx 27\%$ compared to AFSL and FSL, respectively. A similar reduction of $\approx 49\%$ and $\approx 58\%$ is observed on the VoD dataset when compared to AFSL and FSL. Note that a reduced network overhead can also lowers network energy consumption. This reduction is attributed to the selective participation of only K devices in each training round, rather than all N devices, which effectively reduces communication overhead. When examining training time, AFSL+ takes longer on the MNIST dataset compared to AFSL and FSL. This is likely due to the selection of clients with lower compute capabilities, also known as "stragglers." Although this results in a longer training time, the overall energy consumption of AFSL+

Table 5.1: Comparing performance to reach a target F1-score or nMean

Dataset	Metric	Algorithm	Num. server aggregation	Energy (Joules)	Overhead (Gbits)	Training time (Secs)
MNIST	F1-score Targeted value 96%	FSL	56↓32.5%	1,314↑40%	7.9↓13.7%	211↑19.8%
		AFSL	83	923	9.2	176
		Random	140↑68.6%	1,993↑115.9%	10.8↑17.3%	508↑188.6%
		IntraCluster	108↑30.1%	1,413↑53.0%	9.9↑7.6%	424↑140.9%
		AFSL+ ($\alpha = 0.0$)	40↓51.8%	723↓21.6%	5.8↓36.6%	324↑84.0%
		AFSL+ ($\alpha = 0.5$)	32↓61.4%	712↓22.8%	5.8↓36.6%	318↑80.6%
		AFSL+ ($\alpha = 1.0$)	48↓42.1%	754↓18.3%	6.0↓34.4%	320↑81.8%
VoD	nMean Targeted value 0.5	FSL	97↓21.7%	2,245↑38.7%	36.0↑24.1%	608↑51.2%
		AFSL	124	1,618	29.0	402
		Random	154↑24.1%	2,323↑43.5%	35.8↑23.4%	550↑36.8%
		IntraCluster	112↓9.6%	1,524↓5.8%	27.8↓4.1%	384↓4.4%
		AFSL+ ($\alpha = 0.0$)	65↓46.0%	1,012↓37.4%	15.1↓47.9%	356↓11.4%
		AFSL+ ($\alpha = 0.5$)	64↓48.3%	1,009↓37.6%	15.0↓48.2%	348↓13.4%
		AFSL+ ($\alpha = 1.0$)	67↓45.9%	1,060↓34.4%	15.1↓47.9%	360↓10.4%

Note: ↓x% means reduction of x% compared to AFSL. ↑x% means increase of x% compared to AFSL.



(a) MNIST Dataset

(b) VoD Dataset

Figure 5.9: Cumulative data transferred per method.

demonstrates the value of strategically selecting such clients. Even though these clients slow down the training process, their energy-efficient usage should not be overlooked, as their contribution helps minimize overall energy consumption and improve training performance.

In contrast, on the VoD dataset, AFSL+ completes training faster than both AFSL and FSL. This suggests that the clustering approach and client selection mechanism used by AFSL+, which balances energy and performance metrics, can achieve its objectives without necessarily prioritizing speed. This finding indicates that faster training does not always correlate with lower energy consumption. As shown in Table 5.1, AFSL+ can meet its energy minimization goals with varying α values, even when training time is not minimized. Ultimately, the compute capacity of selected clients plays a crucial role in determining training time, with faster training possible in some cases (e.g., VoD) and slower in others (e.g., MNIST), depending on the clients chosen.

Moreover, the comparison with alternative client selection strategies further underscores the advantages of AFSL+. Random client selection yields the worst performance overall, while intra-cluster selection, where clients from the same data distribution cluster participate simultaneously, shows inferior results compared to AFSL+. This demonstrates the value of diverse and representative data participation in each round. By intentionally mixing clients from different clusters, AFSL+ ensures broader coverage of the data distribution, which not only improves generalization but also stabilizes convergence. These findings collectively highlight AFSL+ as a robust and adaptive learning framework that intelligently balances energy consumption, model accuracy, and convergence time in heterogeneous environments.

5.5 Conclusion

In this Chapter, we introduced AFSL, an asynchronous federated split learning framework that addresses client heterogeneity and improves training efficiency. Our results show up to 86% faster learning compared to existing methods, with strong performance under non-IID data and dynamic client participation. We further proposed AFSL+, which enhances AFSL with energy-aware client selection, significantly reducing energy consumption and network overhead without compromising model accuracy. Together, these contributions pave the way for more scalable, robust, and energy-efficient distributed learning in real-world network environments.

Chapter 6

Real-Time Network-Aware Compression for Efficient Split Learning in Distributed Systems

The rapid expansion of distributed machine learning, particularly in environments with limited bandwidth and fluctuating network conditions, underscores the need for efficient communication strategies. In this chapter, we propose a novel dynamic compression technique for Split Learning (SL), which adapts in real-time to network fluctuations, optimizing the trade-off between communication overhead and model performance. Unlike conventional approaches, our method dynamically adjusts the compression level based on current network conditions, significantly reducing the amount of data transmitted during both forward and backward propagation without compromising model accuracy. Our technique not only minimizes communication overhead but also serves as an implicit form of regularization, preventing the model from memorizing irrelevant patterns, thereby improving generalization.

6.1 Introduction

Machine learning has become a cornerstone of modern technological advancements, enabling systems to autonomously identify patterns in data and make decisions or predictions. However, traditional cloud-based training methods, which rely on centralizing data for processing, are increasingly inadequate to meet the demands of modern applications. These include the Internet of Things (IoT), smart healthcare, intelligent

transportation, autonomous driving, and natural language processing, where there is a rapid surge in data traffic, a need for massive computing power, strict latency constraints, and the requirement for personalized services [Letaief et al., 2021; Kuang et al., 2024]. Additionally, concerns about protecting users' personal data have made centralized approaches less viable, as data privacy regulations and user expectations increasingly call for secure and distributed alternatives.

In response to these limitations, Distributed Machine Learning (DML) has emerged as a promising solution. DML enables model training across multiple devices or systems while maintaining data privacy and minimizing the need for centralized data processing. This approach is particularly beneficial in communication networks, where data is generated by a diverse set of geographically dispersed devices, from edge devices like smartphones and wearables to base stations and the cloud. By reducing the computational burden on centralized systems and optimizing resource usage, DML is becoming essential for the scalability and efficiency of modern communication infrastructures. For instance, 3GPP Release 18 [3GPP, 2024a] and Release 19 [3GPP, 2024b] for 5G standardization discussed Federated Learning (FL), a prominent example of DML. FL has gained attention for its ability to train models without transferring sensitive data to centralized servers. The key idea of FL is to build a global model by aggregating locally trained models from distributed devices, ensuring data privacy [Bonawitz et al., 2019]. However, FL can be impractical for many IoT devices, which are often resource-constrained and unable to handle the computational load of training large models locally.

To overcome these limitations, Split Learning (SL) [Gupta and Raskar, 2018] has emerged as an effective alternative, especially in environments with limited resources, such as wireless networks and edge computing systems. As illustrated in Figure 6.1, SL divides the model into two parts: one part is trained locally on the client device, while the other part is processed on the server side. This division significantly reduces the computational, storage, and memory demands on client devices, making machine learning feasible for resource-constrained devices. As a result, SL is expected to play a vital role in next-generation networks, such as 6G, which promise ultra-low latency and high throughput [Zhang et al., 2019c; Letaief et al., 2021; Lin et al., 2024a]. By offloading computational tasks and preserving data privacy, SL enables the training of deep neural networks with millions or even billions of parameters directly at the edge, optimizing resource utilization across the network. However, despite its advantages, communication overhead remains a significant challenge in split learning. During the forward and backward passes of training, the client and server exchange data, which can incur substantial communication costs, especially with large-scale models or in

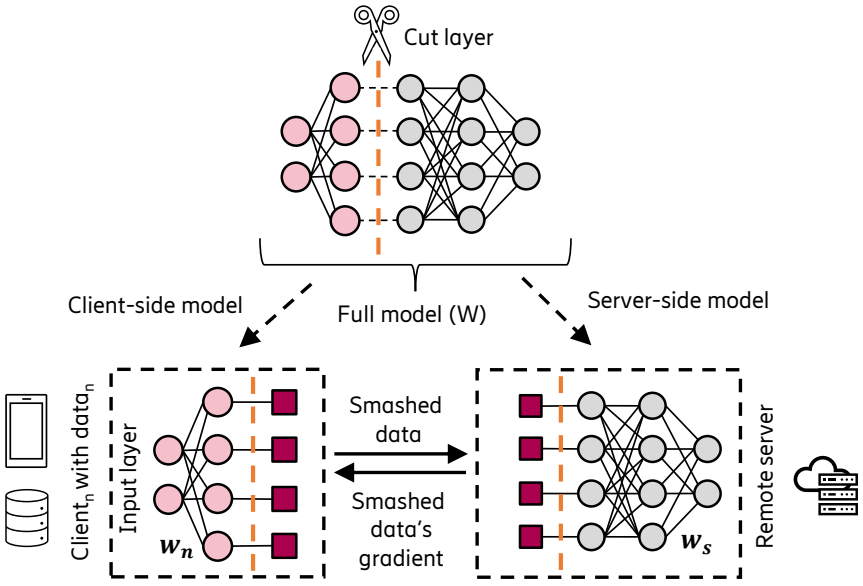


Figure 6.1: SL overview: a cut layer splits the full model into two remote parts known as client-side model and server-side model.

scenarios where network bandwidth is limited. For instance, consider a simple split learning setup where a convolutional network is divided into a client-side and a server-side component. In this setup, the client transmits the output of the first convolutional layer to the server, while the server computes the final loss and gradients. If each client sends a 28×28×1 input image through a convolutional layer (32 filters, 3×3 kernel), the activation data sent to the server would be around 84.5 KB per iteration. Simlilarly, the gradients sent back to the client during the backward pass would also be around 84.5 KB, assuming the data type for each activation and gradient is 4 bytes. For 10,000 iterations, this results in over 1.6 GB of data transfer, even for a relatively small model and dataset (1 image dataset and 1 client). This communication overhead can create bottlenecks, leading to delays and inefficiencies, particularly in environments with fluctuating network conditions and limited bandwidth.

Existing techniques have attempted to mitigate communication costs by employing compression techniques, such as split layer selection [Lin et al., 2024b; Li et al., 2024b; Samikwa et al., 2022], intermediate-data frequency reduction [Chen et al., 2021a; Han et al., 2021; Wang et al., 2022a; Chopra et al., 2023], quantization/sparsification [Oh et al., 2023; Zheng et al., 2023; Yeom et al., 2023; Yuan et al., 2020; Stich et al., 2018], and autoencoders [Ayad et al., 2021] to reduce the amount of data transmitted. While effective to some degree, these methods fail to account for network fluctuations and do not consider dynamically trading accuracy for communication efficiency. In fact, they typically apply a fixed level of compression

throughout the training process, regardless of the actual network conditions. This fixed approach may not be optimal, as the required compression level can fluctuate depending on the current network bandwidth and latency. For example, during periods of high network bandwidth, lower compression levels could preserve model accuracy, while during periods of low bandwidth, higher compression levels might be necessary to ensure efficient communication.

Contributions

In this chapter, we propose a novel approach to optimize communication efficiency in split learning by dynamically adjusting the compression level based on real-time network conditions. The following are the key contributions of our work:

- **Novel Dynamic Compression Technique:** We introduce a novel method that dynamically adjusts the compression level in SL, ensuring that communication overhead is minimized while maintaining controlled and minimal accuracy degradation. This approach continuously monitors network conditions and rapidly adapts the compression strategy accordingly, alleviating the network and improving the overall efficiency of distributed learning processes, particularly during network congestion.

- **No additional learnable parameters:** Unlike many existing methods, our technique does not rely on autoencoders, avoiding the need for additional model parameters in the encoding/decoding phase. This simplifies the learning process and eliminates the complexity associated with training additional components.

- **Support for Forward and Backward Propagation:** The proposed technique seamlessly integrates into both forward and backward propagation phases of split learning. This ensures compatibility with the full training cycle, making it suitable for diverse learning scenarios.

- **Compatibility with Multiple Architectures and SL-Based Techniques:** Our method is compatible with various neural network architectures, including fully connected layers, Convolutional Neural Networks (CNN), and others. This flexibility allows it to adapt to a wide range of machine learning models. Furthermore, our approach demonstrates versatility by being compatible with multiple Split Learning (SL) techniques, such as standard Split Learning (SL) [Gupta and Raskar, 2018], Federated

Split Learning (FSL) [Turina et al., 2021], and our proposed Asynchronous Federated Split Learning (AFSL). This broad adaptability ensures its applicability across diverse distributed learning setups.

- **Experimental Validation in a Real Testbed:** To assess the effectiveness of our method, we conducted experiments on a real testbed comprising IoT devices and an edge server. The results demonstrate the method's ability to enhance communication efficiency while minimizing its impact on accuracy across various SL-based use cases.

- **Extensive Network Performance Analysis:** We perform a thorough analysis of network stability by evaluating key performance indicators (KPIs), including Round Trip Time (RTT), TCP retransmissions, and bandwidth utilization. Our results show that the proposed method maintains stable network performance even under heavy congestion, making it suitable for real-world communication-intensive environments, such as 5G and 6G networks.

6.2 Related Work

To overcome the communication overhead challenge in Split Learning (SL), various communication-efficient SL frameworks have been proposed in the literature. These frameworks aim to enhance SL efficiency by reducing (directly or indirectly) the data transferred in the forward and backward propagation phases during the training process. The proposals can be categorized into four groups based on their core principles: (i) split layer selection, (ii) intermediate-data frequency reduction, (iii) quantization/sparsification, and (iv) autoencoders.

In the first group of related works, client-side computation costs and communication overhead in SL are reduced by strategically selecting the model split point. Several works exploiting this technique have been proposed in the literature and, among them, [Lin et al., 2024b; Li et al., 2024b; Samikwa et al., 2022] stand out by considering the heterogeneous nature of each device's resource capability. In [Lin et al., 2024b], a novel adaptive scheme for Split Federated Learning (SFL) for resource-constrained edge networks, named AdaptSFL. The impact of the model splitting point (MS) and client-side model aggregation (MA) in SFL is investigated and, to minimize the training latency, a novel framework that adaptively controls and optimizes MS and MA based on client resources is proposed. Results indicate the effectiveness of the tailored client-side MA and MS strategy in terms of training time reduction for a given target accuracy. Similarly

to [Lin et al., 2024b], the authors in [Li et al., 2024b] design an adaptive scheme for split learning (ASL) that dynamically selects the split point for each device based on its resource capability and channel condition variations. An online optimization algorithm is formulated in order to minimize the average training latency subject to energy consumption constraint. Simulation results demonstrate the effectiveness of the framework when compared to existing SL schemes. Likewise, [Samikwa et al., 2022] introduced an Adaptive Resource Aware Split Learning (ARES) scheme for efficient model training in IoT environments. By strategically implementing device-specific split points to adapt to dynamic changes in network throughput and computing resources, the framework is designed to accelerate training processes and save energy on constrained IoT devices, while addressing the challenges posed by stragglers during training. Moreover, it carefully considers application constraints and objectives to find a balance between minimizing energy consumption and training time, thus offering a comprehensive solution to enhance model training efficiency in IoT systems.

The second group of related works primarily focus on reducing the frequency of intermediate data being transferred via different approaches. In [Chen et al., 2021a], a method to reduce update frequency by updating the client-side model only when the loss decrease exceeds a set threshold is proposed. This eliminates the need for activation/gradient exchange if the client-side model remains unchanged. Additionally, to reduce communication overhead, activations and gradients are quantized from 32-bit to 8-bit floating point with minimal loss of accuracy. However, this reliance on threshold loss may slow convergence and responsiveness to data changes, which can hinder overall learning progress. The effectiveness of the approach may also vary based on the chosen threshold and dataset characteristics, reducing its robustness across different domains. In [Han et al., 2021], the authors proposed a tailored approach for split learning, introducing alternative local loss functions. They developed an algorithm that integrates an auxiliary network within the client-side architecture to generate predictions locally. These predictions are then utilized to calculate local client-side losses, facilitating updates without requiring backpropagated signals from the server. Concurrently, while client-side models are updated using local errors, the main server performs server-side model updates in parallel based on received activations, which are subsequently aggregated. However, a potential limitation of this approach is that relying solely on local client-side losses for updates may lead to divergent model behavior, as these updates may not fully align with the global optimization objectives. In [Wang et al., 2022a], the authors proposed FedLite to address the high communication costs associated with Split Learning schemes. The framework clusters the intermediate features of training data based on mini-batch

similarities using an algorithm designed with product quantization, communicating only the cluster centroids to the server. To mitigate the impact of clustering on accuracy, a gradient correction method is applied. Experimental results demonstrate a significant improvement in compression rate while minimizing accuracy loss. In [Chopra et al., 2023], a new version of SL defined as Adaptive SL (AdaSplit) is introduced. By eliminating the transmission of the gradients from the server to the client and updating only sparse partitions of the server model, the communication overhead and server computational cost are reduced. Thus, trough hyper-parameters regulation, the proposed scheme is able to adapt to variable resource budgets among heterogeneous clients.

In the third group of related works [Oh et al., 2023; Zheng et al., 2023; Yeom et al., 2023; Yuan et al., 2020; Stich et al., 2018], the communication overhead required for transmitting intermediate data in SL is reduced via dropout and/or quantization techniques. In [Oh et al., 2023], the authors introduced adaptive feature-wise dropout, a method enabling the identification of activations that significantly impact model performance. This technique correlates activations with feature importance, determining which should be transmitted and received accordingly. Additionally, non-dropped intermediate feature and gradient vectors are quantized using adaptive levels based on vector ranges, further reducing data transmission. This approach substantially mitigates both uplink and downlink communication overheads. However, its main limitation lies in the requirement to ascertain feature importance, typically necessitating iterative iterations. Moreover, if the model is already trained with essential features, the benefits may be marginal. In [Zheng et al., 2023], the authors present RandTopk, a novel approach inspired by Top-k sparsification, which retains the top-k elements in a vector based on their magnitudes, while setting all other elements to zero. RandTopk introduces randomness to the selection process, providing non-top-k neurons with an opportunity to be chosen. This method exhibits advantages over traditional Top-k sparsification in terms of both convergence and generalization. Its challenge is the difficulty to select k. In [Yeom et al., 2023], an approach that employs channel coding to enhance data reliability during transmission is proposed. It adds redundant bits to the original data, enabling error correction at the receiver end without necessitating packet re-transmission. Unlike conventional methods, which repeatedly send entire packets until successful transmission, this approach transmits the coded packet only once, reducing traffic overhead. A key challenge of this method is determining the appropriate number of redundant bits to add for optimized error correction capability. If not properly selected, this can degrade accuracy and overall performance. Furthermore, implementing and optimizing the channel coding scheme for error correction can introduce complexity, both in terms of

algorithm design and computational overhead. In [Yuan et al., 2020], a new framework to reduce the impact of the limited computational and communication capacity of IoT healthcare devices is proposed. Extending the idea of sparsification of gradients presented in [Stich et al., 2018], the authors further reduce the network traffic by sparsifying the top k elements at each iteration for both activations (forward propagation) and the corresponding gradients (backward propagation).

Regarding the fourth group of related works [Ayad et al., 2021], a split learning framework incorporating an autoencoder neural network between client and server nodes is introduced. This integration aims to minimize data transmission during the forward pass, alongside an adaptive loss threshold mechanism that regulates gradient matrix exchanges during backward propagation, further reducing update transmissions. Compared to the original split learning system, this modified approach reduces both communication and computational overhead. However, a notable drawback is its lack of modularity, as the selection and training of the autoencoder architecture are problem-dependent and require separate training. This process can require significant time and effort, with no guarantees regarding the effectiveness of the autoencoder.

Concluding Remarks

In this work, unlike existing studies, we propose a novel adaptive encoder-decoder framework, adaptable to any SL technique, that compress both uplink and downlink data transfers in SL. The framework is lightweight and designed to reduce the network overhead by dynamically adjusting compression levels based on real-time network conditions. By monitoring the channel state, our proposal significantly reduces the impact of SL training on the network, reliving network congestion when needed. Table 6.1 summarizes the main characteristics of each related work and compares them with our proposal. The following section provides a detailed description of our approach.

Table 6.1: Literature review comparison

Category	Related works	Reduce bandwidth consumption in SL	Uplink compression scheme	Downlink compression scheme	Dynamic compression scheme	No additional neural network parameters	Channel condition awareness	Layers agnostic
Split layer selection reduction	[Lin et al., 2024b]	✓	x	x	N/A	✓	x	✓
	[Li et al., 2024b]	✓	x	x	N/A	✓	✓	✓
	[Samikwa et al., 2022]	✓	x	x	N/A	✓	✓	✓
Intermediate data frequency reduction	[Chen et al., 2021a]	✓	x	x	N/A	✓	✓	✓
	[Han et al., 2021]	✓	x	x	N/A	✓	✓	✓
	[Wang et al., 2022a]	✓	x	✓	x	x	x	x
	[Chopra et al., 2023]	✓	x	x	N/A	✓	x	✓
Quantization/ sparsification	[Oh et al., 2023]	✓	✓	✓	x	✓	x	✓
	[Yeom et al., 2023]	✓	✓	x	✓	✓	✓	✓
	[Zheng et al., 2023]	✓	✓	✓	x	✓	x	✓
Autoencoder	[Ayad et al., 2021]	✓	✓	✓	✓	x	x	x
	Our proposal	✓	✓	✓	✓	✓	✓	✓

N/A: Not available in the proposed solution

6.3 Adaptive Encoder-Decoder Framework

6.3.1 System Model

A SL-based scenario consists of a tail node (e.g., a remote edge server) and N heterogeneous head nodes (e.g., devices or clients), each connected to the tail node through a specified network. These two parts can be technically described as follows:

- **Head Nodes or Client Devices**: N heterogeneous devices collaborate with the edge server to train a deep neural network model in a distributed manner. Each device n is deployed with a device-side model, denoted by $\mathbf{W}_n$. Devices may have varying and limited computational capabilities, as well as differing qualities and network capacity of communication networks. Each device n has its local data set $D_n = \{\mathbf{x}_n^i, y_n^i\}_{i=1}^{|D_n|}$, and the data sets are distributed in a non-IID manner across devices. Here, $\mathbf{x}_n^i \in \mathbb{R}^{d\times 1}$ and $y_n^i \in \mathbb{R}^{1\times 1}$ represent an input data sample and its corresponding label, respectively, where d denotes the dimension of the input data sample.

- **Tail Node or Edge Server**: At the edge server, the server-side model denoted by $\mathbf{W}_s$, continue the training with the smashed data coming from the client side and send back the gradients to the clients to conclude each training iteration.

Therefore, in SL, the server and devices collaboratively train the global AI model $\mathbf{W} = \{\mathbf{W}_n; \mathbf{W}_s\}$ without sharing either the local data at devices D_n or the client-side model $\mathbf{W}_n$.

On the client side, the first part of the model $\mathbf{W}_n$ which is usually composed of a few layers, uses its local data D_n to compute the intermediate feature data (also known as smashed data), $\mathbf{I}_n$ defined as,

$$\mathbf{I}_n = h(\mathbf{W}_n, \mathbf{x}_n) \in \mathbb{R}^{k\times B}, \tag{6.1}$$

where h is the device-side function that maps the input data to the feature space, $\mathbf{x}_n \in \mathbb{R}^{d\times B}$ the mini-batch input features, k the dimension of intermediate features for one input data and B the batch size. The intermediate feature data $\mathbf{I}_n$ along with the corresponding labels y_n are then sent to the server. At the server-side, the smashed data $\mathbf{I}_n$ is used as input to complete the forward propagation expressed as,

$$\hat{y}_n = g(\mathbf{W}_s, \mathbf{I}_n) \in \mathbb{R}^{1\times B}, \tag{6.2}$$

where g is the server-side function that maps intermediate features $\mathbf{I}_n$ to the output vector $\hat{y}_n$. During the back-propagation, the intermediate gradients of the same size $k \times B$ as the intermediate feature data are computed at the server and sent back to the client-side for parameter update, completing one client iteration.

Implementing the SL framework faces a notable hurdle due to the considerable communication overhead required to transmit the intermediate data features and the gradients. This challenge intensifies with larger mini-batch sizes and dimensions of intermediate features for input data. To improve the practicality of SL, it is crucial to minimize the communication overhead involved in transmitting both intermediate feature data and gradients. Additionally, it is important to enable training and/or deployment of an SL-based neural network model even over a poor communication channel, rather than being unable to generate a neural network at all. The following sections describe our proposed solution.

6.3.2 Network Quality based Adaptive Feature and Gradient Compression

A common limitation observed in the state-of-the-art studies discussed in Section 6.2 is the inability to account for the dynamic nature of the communication channel, which can change over time and impact the volume of information that can be transmitted. This section presents the framework we propose to dynamically adapt the size of the data transfer according to the state of the network. The primary goal is to consistently minimize communication overhead, which enhances network stability by reducing congestion and latency, and improves efficiency by optimizing bandwidth usage. Our approach leverages performance metrics of the communication channel to determine optimized data compression levels for transmission between clients and servers. Specifically, the framework adapts the compression level based on the channel's capability to reliably transmit data between different parts of the split neural network. When the channel quality is high, indicating strong Key Performance Indicators (KPIs), less compression is applied. Conversely, when the channel quality is lower, indicating weaker KPIs, more aggressive compression is employed to ensure efficient data transmission and alleviate the channel. Notably, our method does not rely on autoencoders, eliminating the need for learning additional parameters.

Moreover, our proposal is equally effective for both forward (uplink direction) and backward propagation (downlink direction) during training processes. It seamlessly integrates into the training of split ML models and supports the inference phase for generating outputs from input data. An additional benefit of our framework is its versatility across various model architectures, enhancing flexibility and simplifying integration into existing architectures (e.g., fully connected layer, CNN, etc.). Furthermore, the framework

is designed to accommodate different split learning approaches, including SL [Gupta and Raskar, 2018], SFL [Thapa et al., 2022] PSL [Jeon and Kim, 2020], FSL [Turina et al., 2021], our proposed AFSL [Albuquerque et al., 2024] and others. This adaptability ensures broad applicability and robust performance across different distributed learning scenarios.

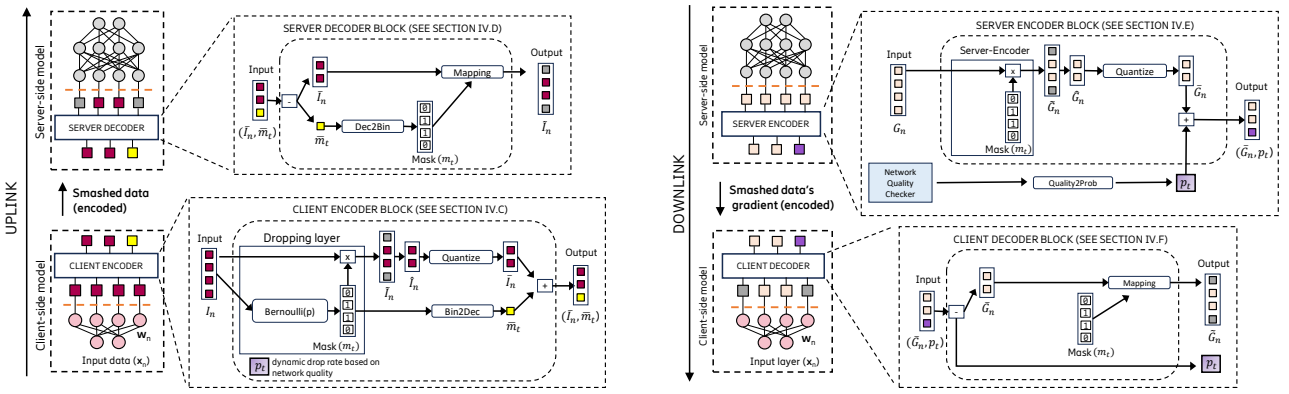
As illustrated in Figure 6.2, our proposed solution consists of five key components. *1) Network quality checker:* this component monitors the network state and provides data to adapt the encoder parameters at both the client and server nodes. *2) Client encoder:* it transforms the intermediate feature data $\mathbf{I}_n \in \mathbb{R}^{k \times B}$ into a reduced latent vector $\bar{\mathbf{I}}_n \in \mathbb{R}^{\bar{k} \times B}$ (where $\bar{k} < k$), which is then transmitted during the forward phase (uplink). *3) Server-decoder:* this component reconstructs the transmitted data, referred to as $\tilde{\mathbf{I}}_n$. *4) Server encoder:* in the backward phase, it transforms the gradient into a reduced gradient vector for transmission. *5) Client decoder:* it reshapes the gradient back to its original dimensions. Each of these components is described in more detail in the following/corresponding sections.

Network Quality Checker

This component is designed to assess the state of the network by monitoring the quality and performance of the communication channel between clients and the server. It evaluates performance by monitoring one or more network Key Performance Indicators (KPIs), such as packet loss, latency, bandwidth utilization, signal quality, Reference Signal Received Power (RSRP), Reference Signal Received Quality (RSRQ), interference, pathloss, Block Error Rate (BER), round trip time, and others. It is important to note that the network quality checker can be applied to any type of communication channel (e.g., wired or wireless), and the selected KPIs may vary depending on the channel type.

At each iteration t, the *network quality checker* captures the current state of the network. This information is then used by *quality2Prob* to calculate the reduction factor p_t. When the network state changes, a new reduction factor is generated and sent to the client to adjust its compression rate accordingly. If the network condition remains unchanged, no new reduction factor is transmitted, and the client maintains its current compression rate. This module could be implemented as a trained machine learning model or any network management tool capable of assessing network states. It can be deployed at any point in the network (e.g., client/head node or server/tail node). Without loss of generality, we assume it is located at the server node in our work, as depicted in Figure 6.2(b).

The *quality2Prob* module is designed to ensure that the generated reduction factor p_t accommodates



(b) [Downlink compression.

Figure 6.2: Proposed encoder-decoder scheme for communication efficient SL-based approaches.

poor network conditions (e.g., highly congested networks) while maintaining an acceptable level of accuracy. This is achieved by limiting compression up to a certain threshold. This design ensures that our solution can effectively train a model even under challenging network conditions without sacrificing accuracy.

Client Encoder

The proposed encoder consists of three primary layers: the dropping layer, the quantization layer, and the masking layer. Each of these layers contributes to generating the data that will be transmitted to the network.

Dropping Layer

We propose a dynamic compression approach that probabilistically drops intermediate feature data based on real-time network conditions (e.g., excellent, good, average, poor). Given the network performance metrics measured by the *network quality checker*, the data reduction factor p_t, i.e., the compression level will be determined. This data reduction factor represents the probability of dropping the output of any particular neuron in the cut layer of the neural network client part. As such, the data reduction factor p_t can be a value between 0 and 1, with higher values in that range indicating a higher likelihood of dropping the output of a neuron when performing compression.

At each iteration t, given the reduction factor p_t, this layer generates a vector of drop probabilities using an independent Bernoulli distribution. This allows for the sampling of the binary mask m_t, i.e., a new vector containing only zeros and ones. By generating the mask in this manner, the neural network is effectively regularized by introducing noise and preventing co-adaptation of neurons, thus improving the generalization

performance of the trained neural network. This mask is then multiplied element-wise with the intermediate feature $\mathbf{I}_n$ to get the final result returned by the dropping layer. The retained activations are scaled up by multiplying them by $\frac{1}{1-p_t}$ to ensure that the expected value of the scaled feature vector remains the same as that of the original feature vector [Srivastava et al., 2014]. The output of this layer $\tilde{\mathbf{I}}_n$ is represented as follows:

$$\tilde{\mathbf{I}}_n = \frac{\mathbf{I}_n \odot \mathbf{m}_t}{1 - p_t}, \qquad b_{t_i} \sim \textit{Bernouilli}(q_t), \tag{6.3}$$

where $\odot$ denotes an element-wise product, p_t is the reduction factor (drop rate hyper-parameter), $q_t = 1 - p_t$ is the keep rate hyperparameter and $m_t \in \mathbb{R}^{k \times 1}$ is a vector of independent Bernoulli random variables b_{t_i}, each of which has a probability q_t of being 1.

All zero-out features of $\tilde{\mathbf{I}}_n$ will be removed, thus reducing the size of the data to be transmitted. Let $\hat{I}_n \in \mathbb{R}^{\bar{k} \times b}$ denote the reduced $\tilde{\mathbf{I}}_n$, with $\bar{k}$ the new feature size.

Table 6.2: Overview of adopted datasets specifications.

Use case	Task type	Target object	Dataset		Evaluation Metric	Acceptable performance drop
			D_x	D_y		
Use case 1: MNIST [LeCun and C, 2010]	classification	Images (digits)	784	10	F1score	3.0 %
Use case 2: 5G [Rao et al., 2022]	classification	5G E2E RTT violation	495	2	F1score	0.5 %
Use case 3: IoT [Yan, 2018]	Regression	IoT network RTT	14	1	nMeanAE	0.04 %
Use case 4: VoD [Samani, 2023]	Regression	Network read avg. delay	197	1	nMeanAE	0.3 %

Quantization Layer

On top of dropping out some features, we propose to further reduce the communication overhead by quantizing $\hat{I}_n = \{\hat{I}_{n_i}\}_{i=1}^{\bar{k}}$, the output of the dropping layer. Any quantization approach can be use (e.g., fixed-point quantization, clipping, uniform quantization, stochastic quantization [Howard, 2017; Jacob et al., 2018; McMahan et al., 2017; Alistarh et al., 2017]) as long as it maintains acceptable performance. For simplicity, we used a floating point quantization where each f-bit floating-point representation of $\hat{I}_{n_i}, i \in \{1, 2, \dots \bar{k}\}$ will be quantized to a $\bar{f}$-bit floating point representation. Indeed, considering the general representation of the floating-point $\hat{I}_{n_i}$ given by

$$\hat{I}_{n_i} = (-1)^s \times 1.F \times 2^e, \tag{6.4}$$

where s represents the sign bit, F the fractionnal part of the mantissa represented with f-bits, and e the

exponent. Each quantized $\bar{f}$-bit floating point $\bar{I}_{n_i}, i \in \{1, 2, \dots \bar{k}\}$, of $\bar{I}_n$ can be expressed as

$$\bar{I}_{n_i} = (-1)^{\bar{s}} \times 1.\bar{F} \times 2^{\bar{e}}, \tag{6.5}$$

where $\bar{s}$ represents the sign bit directly taken from the original sign bit s, $\bar{F}$ the fractional part of the mantissa represented with $\bar{f}$-bits, and $\bar{e}$ the exponent. The quantization process involves rounding the mantissa and adjusting the exponent accordingly to fit the $\bar{f}$-bit format. Hence, these quantized parameters are found based on the parameters of $\hat{I}_{n_i}$ as follows:

$$\bar{F} = Round(F, \bar{p}), \qquad \bar{e} = Round(e + \bar{b} - b), \tag{6.6}$$

where b is the bias for the f-bit representation, $\bar{b}$ the bias for the $\bar{f}$-bit representation and $\bar{p}$ is the number of bits allocated to the mantissa in the $\bar{f}$-bit floating point representation. It is important to specify that the values of b, $\bar{b}$, and $\bar{p}$ are standardized in formats like for example IEEE 754, which is widely used for floating point representation.

Binary to Decimal Converter Layer

As an optional step, at each iteration t, the mask $m_t = [\beta_{t_i}]_{i=1}^{k}, \beta_{t_i} \in \{0, 1\}$, consisting of zeros and ones, can be transformed into a more compact representation that requires fewer bits for transmission. Here, β_{t_i} denotes the binary digit at position i in the mask m_t.

We suggest employing a binary-to-decimal converter for this purpose, although any other converter that reduces data size could be utilized. The converted mask is denoted by:

$$\bar{m}_t = \sum_{i=1}^{k} \beta_{t_i}.2^{k-i}. \tag{6.7}$$

Hence, the encoder will send to the network the compressed intermediate latent feature vectors $\bar{\mathbf{I}}_n$, along with the corresponding mask $\bar{m}_t$ that uses fewer bits than m_t. Figure 6.2(a) illustrates the client encoder block.

Server-Decoder

Upon receiving $\bar{\mathbf{I}}_n$ and $\bar{m}_t$, the decoder layer will generate $\tilde{\mathbf{I}}_n$, which is the reconstruction of the output from the dropping layer. This reconstructed output will be employed to complete the forward pass at the server level. Figure 6.2(a) presents the server-decoder architecture consisting of two main layers: the decimal-to-binary converter layer and the mapping layer.

Decimal to Binary Converter Layer

This layer is responsible for transforming the received decimal mask $\bar{m}_t$ into its corresponding binary vector m_t for utilization in the mapping layer. This conversion involves iteratively dividing the decimal mask $\bar{m}_t$ by 2 and recording the remainder. The binary representation is then constructed by reading the remainders in reverse order, resulting in the reconstructed binary mask m_t.

Mapping Layer

The mapping layer is responsible for reconstructing the final intermediate feature $\tilde{\mathbf{I}}_n$ from the latent $\bar{\mathbf{I}}_n$ and the mask m_t. It uses the binary mask m_t and the intermediate latent feature $\bar{\mathbf{I}}_n$ to generate $\tilde{\mathbf{I}}_n$ which is employed to complete the server forward pass.

Server-Encoder

The inclusion of this layer is contingent upon its influence on the overall training process. Given the known mask m_t from the forward pass and the computed gradients $\mathbf{G}_n$, this layer, similar to the client-encoder, selectively drops out certain gradient units. This results in a compressed gradient $\hat{\mathbf{G}}_n$, which is further compressed to $\bar{\mathbf{G}}_n$ after gradient quantization. The compressed gradient $\bar{\mathbf{G}}_n$, along with the reduction factor p_t (if any), generated by the network quality checker, is then transmitted to the client. Figure 6.2(b) illustrates the server encoder block operation.

Client-Decoder

This layer receives the compressed gradient $\bar{\mathbf{G}}_n$ from the server, reshaping it to its original size $\tilde{\mathbf{G}}_n$ if the server encoder was employed. Additionally, it extracts any associated reduction factor (dropping probability) p_t that will be utilized in the subsequent iteration. Figure 6.2(b) illustrates the client decoder block.

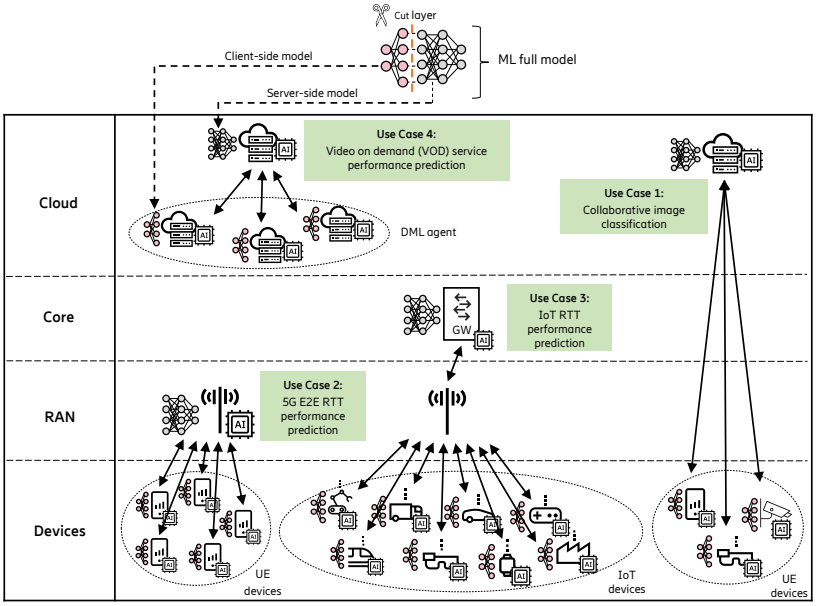


Figure 6.3: Different use cases with machine learning integration across future communication network layers, from devices to cloud.

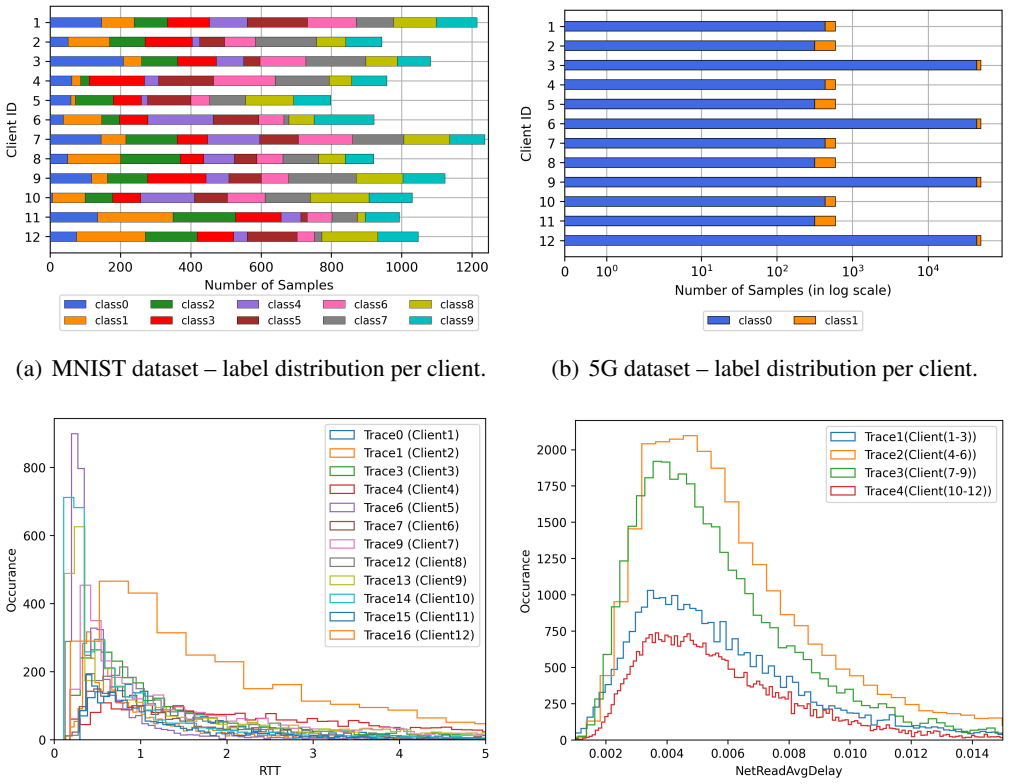
Moving forward, we will discuss the datasets and testbed setup used to evaluate the performance of our approach.

6.4 Datasets and Testbed Setup

To evaluate the performance of our approach in various split learning-based scenarios, we selected four datasets with distinct data distributions and prediction tasks. The following sections provide a detailed description of each dataset and the environmental testbed setup.

6.4.1 Datasets

In this section, we present the four distinct use cases employed to evaluate the proposed approach. These use cases span a variety of domains in machine learning, ranging from collaborative learning in image classification to performance prediction in networking environments. Figure 6.3 illustrates each use case along with its respective dataset, emphasizing its relevance and significance to the overarching goals of the study. The following sections provide a detailed description of each dataset and Table 6.2 summarizes their main characteristics.



(a) MNIST dataset – label distribution per client.

(b) 5G dataset – label distribution per client.

(c) IoT dataset – target value distribution per client.

(d) VoD dataset – target value distribution per client.

Figure 6.4: Data distribution of the adopted datasets.

Use Case 1 – User Equipment (UE) Collaborative Image Classification on MNIST

The MNIST dataset [LeCun and C, 2010] is a widely recognized benchmark in the machine learning community, comprising 70,000 grayscale images (28x28 pixels each) of handwritten digits from 10 classes (digits 0–9). In the first use case, we focus on collaborative image classification, where the objective is to train a machine learning model to classify these handwritten digits in a distributed manner. Specifically, we simulate this task within the context of a network of user equipment (UE), where each device participates in a collaborative training process without sharing raw data. In this scenario, each UE is responsible for classifying a local subset of the MNIST images. Importantly, the dataset is partitioned in a non-IID (non-independent and identically distributed) manner, meaning that the number of data points and class proportions are unbalanced across UEs. To model this, we partition the data for each user i according to a normal distribution $|N(\mu, \sigma^2)|$ for each class. The mean μ is empirically set to 100, and the standard deviation σ is set to 0.5 μ, introducing variability in the number of samples and class distributions per user. This results in a more realistic scenario where each UE may encounter different distributions of classes, reflecting

real-world variations in data. For better visualization, Figure 6.4(a) shows the data distribution for a sample of 12 UEs. Despite the non-IID distribution during training, it is important to note that the evaluation phase uses the test set, which includes all 10 class labels uniformly. This ensures that the model is tested on a comprehensive set of examples, allowing for a fair assessment of its classification accuracy.

Use Case 2 – End-to-End (E2E) Round-Trip-Time (RTT) Violation Prediction on Data Traces from a 5G-mmWave Testbed

Round-Trip Time (RTT) is a crucial performance metric in networking, significantly impacting user experience in latency-sensitive applications such as video streaming, online gaming, and real-time communications. In this use case, we focus on predicting the end-to-end (e2e) RTT violation in a 5G millimeter-wave (mmWave) network. The goal is to estimate the RTT violation based on various network parameters, including network configuration, user equipment (UE) position, and other contextual factors. The dataset used in this use case consists of RTT measurements obtained from a real-world 5G mmWave testbed, designed to simulate a 5G non-standalone (NSA) system [Rao et al., 2022]. The testbed includes multiple mobile devices and a base station, operating under diverse network conditions. These conditions encompass varying traffic loads and mobility patterns. For example, the network can either generate a constant bit-rate uplink traffic load or operate without it, and the UE can either be stationary or moving. RTT measurements are captured using ICMP ping [Postel, 1981] with a packet size of 1400 bytes, recorded every 10 ms. The dataset includes not only the RTT values experienced by the UE but also approximately 200 additional metrics and events related to various aspects of the network, such as the analogue beamforming function, uplink and downlink events, and other key network performance indicators. From these raw measurements, 495 features were derived by applying different averaging intervals to both the RTT values and the associated metrics. To determine whether RTT violation will occur in the near future, using high-frequency metrics measured at the base station, the prediction of the RTT is modeled as a binary classification problem, with class 0 being the normal RTT behaviour, and class 1 being an RTT violation. Indeed, the RTT for each sample is classified as either ”low” (normal RTT) or ”high” (RTT violation) based on a pre-defined threshold. This threshold serves as a critical decision point, and each RTT sample is labeled accordingly, depending on whether its value is above or below the threshold. Figure 6.4(b) illustrates the data distribution of the RTT values in the dataset, providing a visual representation of the balance between ”low” and ”high” RTT instances.

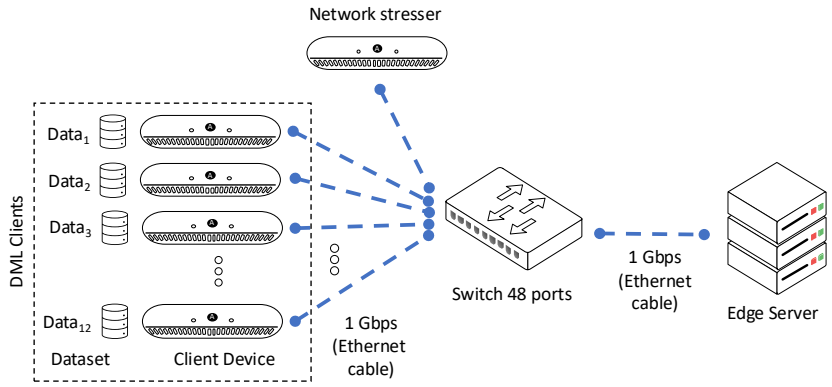
Use case 3 – Round-Trip-Time (RTT) Prediction on Data Traces from an IoT Testbed

5G and the forthcoming 6G networks are set to enable the proliferation of massive IoT applications, which are expected to become ubiquitous, seamlessly integrating into various aspects of daily life and industry [Li et al., 2018b]. Accurate prediction of Round-Trip Time (RTT) in IoT networks is critical for enhancing the reliability and efficiency of IoT-based applications, such as smart home systems, industrial IoT, and health monitoring solutions. By optimizing network performance and reducing latency, RTT prediction can significantly improve user experience and the operational effectiveness of these applications. In this use case, we leverage traces generated from an IoT testbed developed at Uppsala University [Schuß et al., 2017] to predict the RTT between an IoT device and an IoT gateway. The testbed comprises 18 Tmote Sky Motes [1], a popular platform for low-power wireless communication. One mote functions as a Two-Way Active Measurement Protocol (TWAMP) Two-Way Active Measurement Protocol (TWAMP) controller (acting as the server during training) [Hedayat et al., 2008], while the remaining 17 motes serve as reflectors (acting as clients during training) [Flinta et al., 2020]. The dataset contains a variety of network statistics, including Routing Protocol for Low-power and lossy networks (RPL) rank, Received Signal Strength Indicator (RSSI), neighbor count within radio reach, Link Quality Indicator (LQI), CPU usage, and transmitted packets, which are used as features for RTT prediction, resulting in a total of 14 features. These features capture key aspects of the IoT network's operational environment, allowing the model to learn the relationships between network conditions and RTT. Figure 6.4(c) illustrates the distribution of the dataset, providing a visual representation of the RTT values between IoT devices and the IoT controller.

Use case 4 – Service Performance Prediction on Data Traces Collected from a Data Center (DC)

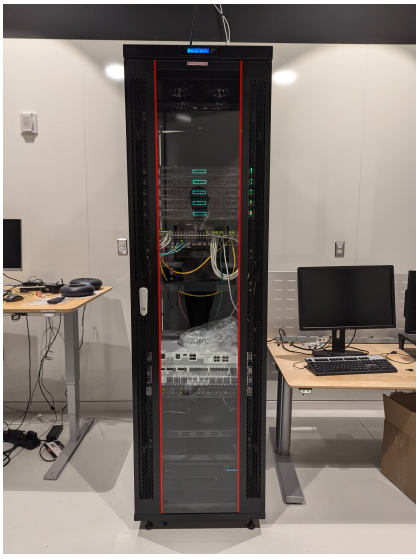
Accurate prediction of service performance is critical for proactive resource management and load balancing in data centers, as well as in 5G and beyond networks, particularly in environments with fluctuating workloads and high computational demands. This use case focuses on predicting service level metrics (SLMs) for clients, using high-frequency traces collected from servers and networking equipment within the data center. These traces provide a rich set of features, enabling the prediction of performance across various services hosted on the data center infrastructure. The dataset used in this work was generated through experiments conducted in a testbed at KTH University [Yanggratoke et al., 2018], which includes a server cluster

[1] https://docs.contiki-ng.org/en/develop/doc/platforms/sky.html



(a) Schematic of the testbed.

(b) Client server cluster connected to the edge server.



(c) Rack with the edge server.

Figure 6.5: Overview of the testbed used for the evaluation

and six client machines. The traces [2] contain approximately 1700 features, such as CPU utilization per core, memory usage, and disk I/O, collected from the server cluster, while service level metrics are gathered from the client machines. Two services are deployed in the testbed: Video-on-Demand (VoD) and a Key-Value (KV) store (database). For our experiments, we focus on the VoD service, using the net value of the average read delay on the client as the service level metric, with a total of 182 features. Figure 6.4(d) illustrates the distribution of the dataset.

[2] https://www.kaggle.com/datasets/jaliltaghia/datatraces-from-a-data-center-testbed.

Table 6.3: Neural network model architectures

	Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Output layer	Optimizer & Loss	Cut Layer
MNIST	Conv2D,6 filters 5x5, Stride 1, ReLU, AvgPool2D, 2x2	Conv2D, 16 filters 5x5, stride 1, ReLU, AvgPool2D, 2x2	Dense, 120 units, ReLU	Dense, 84 units, ReLU		Dense, 10 units, (Classifier)	Adam optimizer learning rate: 1e-3 CrossEntropy Loss Batchsize: 32	Layer 1
5G	Dense, 1200 units, Tanh, Dropout, 0.1	Dense, 1200 units, ReLU, Dropout, 0.1	Dense, 512 units, ReLU Dropout, 0.1	Dense, 64 units, ReLU, Dropout, 0.1		Dense, 2 units, (Classifier)	Adam optimizer Learning rate: 1e-3 CrossEntropy Loss Batchsize: 128	Layer 2
IoT	Dense, 128 units ReLU BatchNorm1D, 128, Dropout, 0.2	Dense, 1200 units, ReLU BatchNorm1D, 1200 Dropout, 0.2	Dense, 128 units, ReLU, BatchNorm1D, 128, Dropout, 0.2	Dense, 64 units, ReLU	Dense, 32 units, ReLU	Dense, 1 unit, (Regressor)	Adam optimizer, Learning rate: 1e-4, SmoothL1Loss Batchsize: 32	Layer 2
VoD	Dense, 256 units, ReLU, BatchNorm1D, 256, Dropout, 0.2	Dense, 128 units, ReLU, BatchNorm1D, 128, Dropout, 0.2	Dense, 64 units, ReLU			Dense, 1 unit, (Regressor)	Adam optimzer Learning rate: 1e-4 SmoothL1Loss Batchsize: 256	Layer 1

Table 6.4: Testbed specifications.

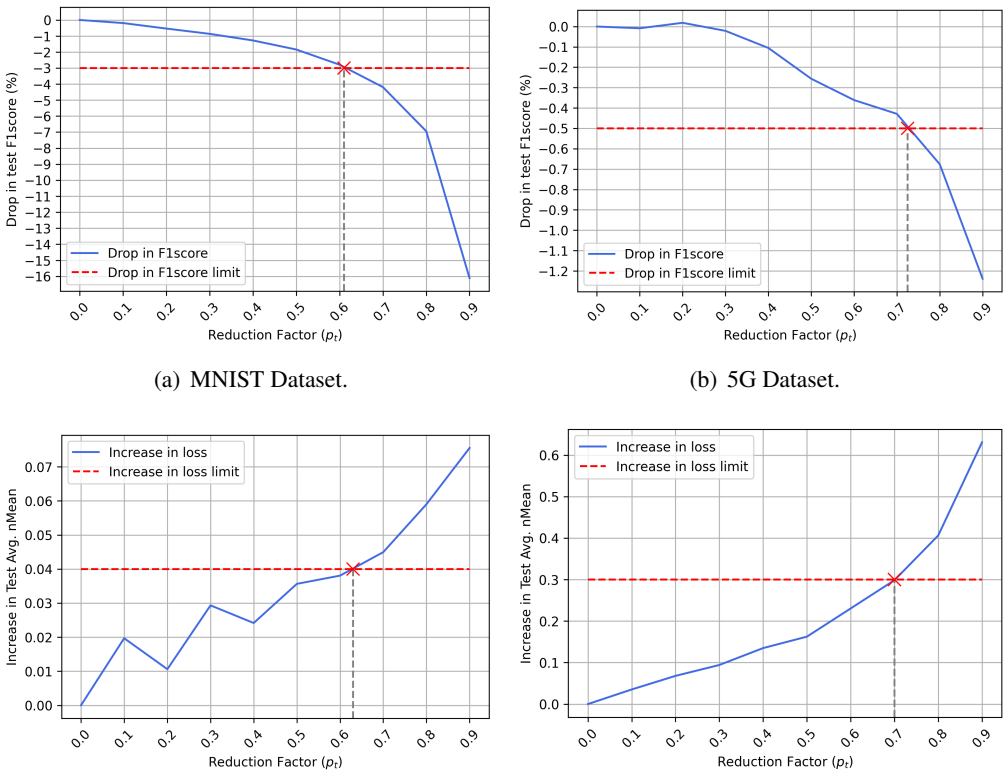
Client Devices	
CPU	AMD Ryzen Embedded V1807B with Radeon Vega Gfx
RAM	900MHz 8GB DDR2
Operating system	Ubuntu 22.04.3 LTS
Edge Server	
CPU	Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz
RAM	2666MHz 130GB DDR4
Operating System	Ubuntu 22.04.4 LTS
Switch	
Model	Summit X440-48t-10G
Switch Ports	48 10/100/1000BASE-T ports
Port capacity	1 Gbps of highdensity copper connectivity

6.4.2 Environmental Testbed Setup

All experiments were conducted on a testbed consisting of multiple client devices (see Table 6.4 for their characteristics) and a single edge server, as illustrated in Figure 6.5. The devices are connected via Ethernet cables through a network switch. Specifications for the testbed devices are provided in Table 6.4. Both the client devices and the edge server are configured with Python 3.7 and PyTorch 2.0 [3]. The following tools were used to perform the necessary measurements throughout the experiments.

- **Variable network conditions:** We used the IPERF tool [Hsu and Kremer, 1998] to stress the 1Gbps switch port interface, simulating varying levels of network congestion to replicate real-world conditions during the training process. Since network conditions are often imperfect, training and inference can occur under fluctuating communication channel qualities. Iperf3[4] is particularly effective for this purpose, as it can generate traffic to reach a specific maximum bandwidth, allowing us to control network load and

[3]https://pytorch.org/
[4]https://iperf.fr/



(a) MNIST Dataset.

(b) 5G Dataset.

(c) IoT Dataset.

(d) VoD Dataset.

Figure 6.6: Selecting the Ada-AFSL dropout threshold for various datasets. The predefined threshold determines the maximum compression level based on the associated reduction in performance. Note that the threshold varies depending on the problem being addressed (dataset).

create different levels of congestion. By adjusting parameters such as packet size, traffic volume, and duration, we can simulate various network conditions, ranging from low bandwidth and high latency to severe packet loss, enabling us to evaluate the robustness of the machine learning model under diverse and realistic network stress scenarios.

- **Network monitoring:** The SNMP protocol was employed to monitor the network, collecting key metrics such as network utilization and traffic in and out at the device, server, and switch levels. This monitored data was then used by the network quality checker to assess network congestion and compute the reduction factor p_t at each iteration.

- **RTT and TCP retransmission measurement:** To assess the impact of our solution on key network performance indicators (KPIs) such as RTT and TCP retransmission, we conducted ICMP nping tests with a constant packet size of 64 bytes, sending packets at 1-second intervals, while continuously measuring RTT.

TCP retransmission was monitored after each training session using the netstat [5] tool. This methodology allowed us to evaluate how our proposed solution, with its adaptive compression mechanism, contributes to maintaining network stability by monitoring its impact on these critical performance metrics.

For each use case, a deep learning model was utilized and partitioned into two distinct parts: one deployed on the client-side and the other on the server-side. In Use Case 1, we selected the LeNet-5 neural network [LeCun et al., 1998] as the global model deployed on both the server and client devices. For the other use cases, DNN models were chosen and fined tune based on their respective datasets. The architectures are detailed in Table 6.3. Note that in SL, a DNN is modeled as a sequence of independently executable layers, not as a single indivisible unit. Our approach is generalizable and can be applied to any sequential DNN composed of independently executable layers.

We evaluate the benefits of the proposed solution by applying the compression scheme to three split learning-based approaches: Vanilla SL (SL) [Gupta and Raskar, 2018], FSL [Turina et al., 2021], AFSL [Albuquerque et al., 2024]

For Use Case 1 and Use Case 2, classification performance is assessed using the F1-score, which combines precision and recall into a single metric by calculating their harmonic mean. The F1-score was chosen due to the imbalanced nature of the datasets in these use cases. A higher F1-score, closer to 1, indicates better performance.

For Use Case 3 and Use Case 4, regression performance is evaluated using the normalized mean absolute error (nMeanAE), which measures the difference between the measured and predicted outputs, given by:

$$nMeanAE = \frac{1}{\bar{y}} \left(\frac{1}{m} \sum_{i=1}^{m} |y_i - \hat{y}_i| \right), \tag{6.8}$$

where $\hat{y}_i$ is the model prediction for the i-th data point, y_i is the true value, and $\bar{y}$ is the mean of the observed values y_i from the test set of size m. Lower $nMeanAE$ values, closer to zero, are preferred.

[5] https://man7.org/linux/man-pages/man8/netstat.8.html

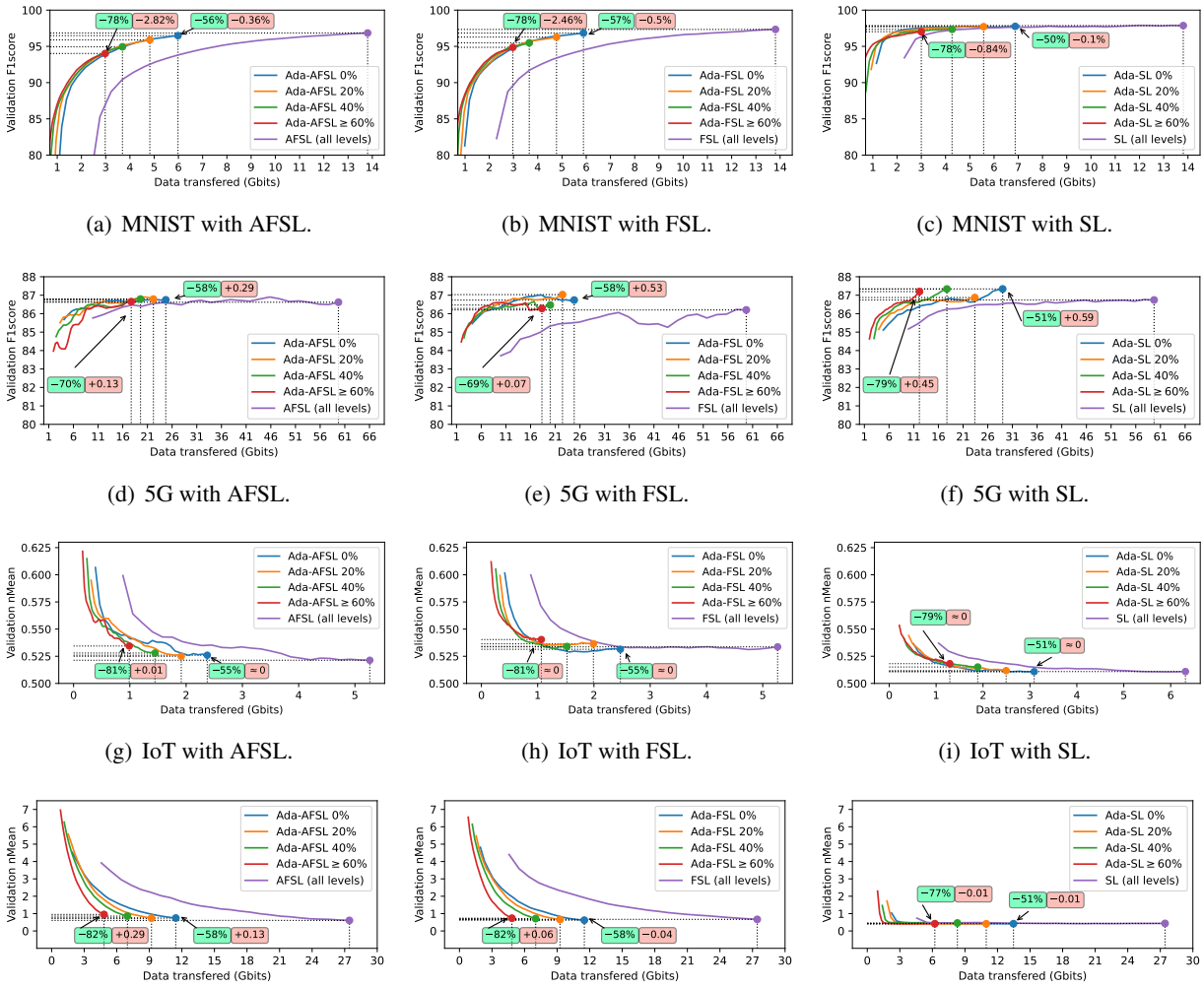
6.5 Performance Evaluation

6.5.1 Selecting the Reduction Threshold

This experiment is done prior to the main training. We aim to find the maximum reduction factor, denoted as $p_{\max}$, that can be applied to the size of the data transferred without causing a performance drop exceeding a predefined threshold, $x\%$. This threshold x is assumed based on the specific use case, reflecting the acceptable level of performance degradation and is given in Table 6.2. Thus, the experiment is structured to assess and quantify the relationship between data reduction and performance, ensuring that the reduction factor remains within the limits that preserve the system's efficiency and effectiveness. In this experiment, we conducted multiple runs with varying reduction factors and recorded the performance to analyze the impact of these reductions. The results for each use case are presented in Figure 6.6.

As expected, both classification and regression use cases exhibit a performance decline as the reduction factor increases. In classification tasks, the F1 score decreases, while in regression tasks, the normalized mean loss (nMean loss) increases. This behavior can be attributed to the loss of information due to data compression, which can degrade the model performance. For the classification tasks, a significant drop in the F1 score is observed when the reduction factor is set to 0.9 on the MNIST dataset ($\approx$ -16%), while the drop is less pronounced for the 5G dataset (around -1.2%). In regression tasks, the increase in nMean loss, which represents performance degradation, is reported as an absolute value. For a reduction factor of 0.9, the nMean loss increases by approximately 0.07 for the IoT dataset and 0.6 for the VoD dataset. These discrepancies in performance degradation across classification and regression tasks can be attributed to the varying representativeness of activations across different datasets. Consequently, compressing the same amount of activations results in varying outcomes depending on the dataset characteristics.

Based on the acceptable performance drop thresholds defined in Table 6.2, we determined the maximum allowable reduction factor $p_{\max}$ for each use case. The computed values for $p_{\max}$ were $p_{\max}^{\text{MNIST}} \approx 0.7$, $p_{\max}^{\text{5G}} \approx 0.7$, $p_{\max}^{\text{IoT}} \approx 0.6$ and $p_{\max}^{\text{VoD}} \approx 0.7$ for the MNIST, 5G, IoT, and VoD datasets, respectively. These $p_{\max}$ values were subsequently used in the main experiment to ensure qualitative outcomes. Specifically, when network conditions degrade, the reduction factor is increased, but it should never exceed the $p_{\max}$ value.



(a) MNIST with AFSL. (b) MNIST with FSL. (c) MNIST with SL.

(d) 5G with AFSL. (e) 5G with FSL. (f) 5G with SL.

(g) IoT with AFSL. (h) IoT with FSL. (i) IoT with SL.

(j) VoD with AFSL. (k) VoD with FSL. (l) VoD with SL.

Figure 6.7: Trade-off between accuracy and data transferred using different datasets and SL-based methods. The green and red box indicate, respectively, the reduction in data transferred and the drop in F1 score (or increase in nMean) for the indicated compression level.

6.5.2 Profiles with Unique Stress Level

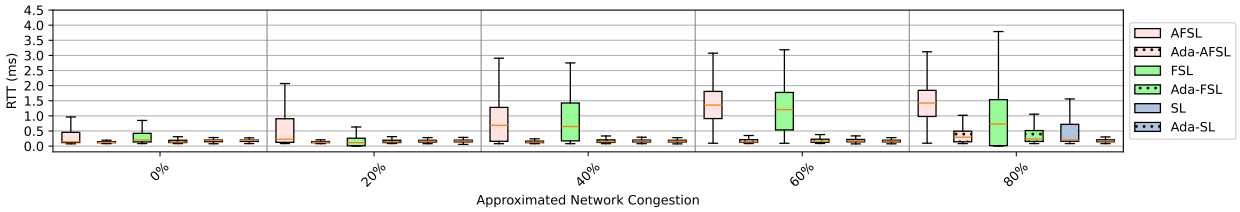
Adaptability to Changing Network Conditions

To assess the benefits of the proposed approach, the training performance (F1 score for classification and nMean for regression) and communication overhead of various split learning methods (SL, FSL, AFSL) under the adaptive scheme (denoted as Ada-SL, Ada-FSL, Ada-AFSL) were compared to their standard counterparts, where no dynamic adaptation is applied. Each approach is set to perform the same number of epochs. Evaluating the approach across multiple split learning methods allows for a broader analysis of its effectiveness and adaptability, as each method has different characteristics and performance trade-offs, ensuring the proposed scheme's robustness in diverse scenarios.

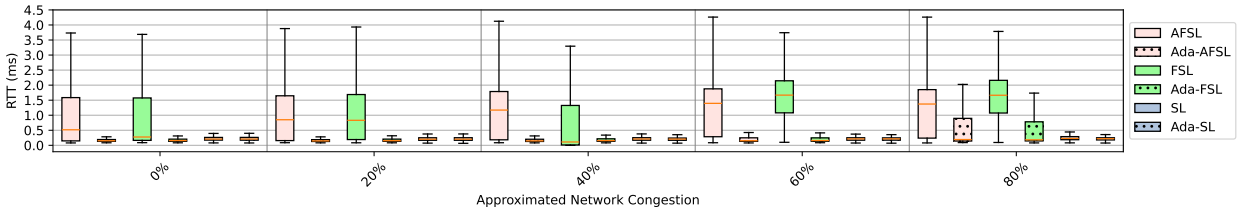
Different scenarios were considered, each with varying network conditions represented by congestion levels. During training, the network was stressed using parameters that simulated different congestion levels. A congestion level of x% indicates that the network was stressed to reach a x% congestion level before training, with this level maintained throughout. For example, 0% indicates no network stress, while 20% represents a network stressed to maintain a 20% congestion level.

For each use case, Figure 6.7 illustrates the training performance and the data transfer volume (calculated as the total communication cost in bits, including the transmission of intermediate feature matrices from devices to the server and gradient matrices from the server to the devices) required for 30 epochs. Since the standard approaches (SL, FSL, AFSL) exhibit similar behavior (showing identical communication costs and relatively consistent training performance) regardless of network congestion, only a single graph is presented, labeled "(all levels)," to illustrate the results.

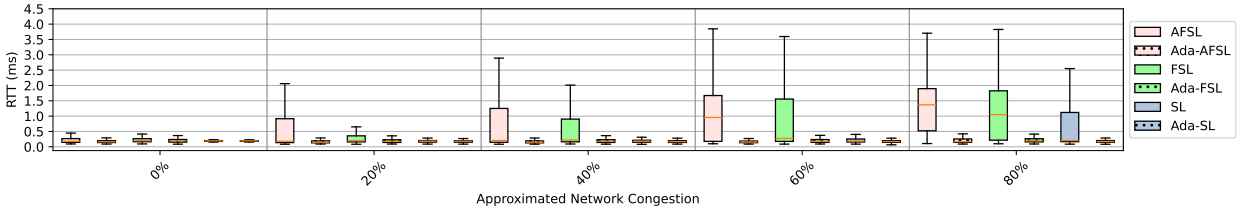
The proposed approach dynamically adjusts communication overhead in response to increasing network congestion, ensuring that performance does not drop below a predefined threshold, as specified in Table 6.2. Specifically, the results show that under low network congestion (i.e., 0%), the proposed adaptive scheme achieves a similar F1 score or nMean as the standard approaches while significantly reducing communication overhead. For a network with 0% congestion, a communication cost reduction of approximately 56%, 58%, 55%, and 58% is observed for MNIST, 5G, IoT, and VoD datasets, respectively, using both AFSL and FSL. Importantly, this reduction in communication costs is achieved with minimal performance loss, consistently less than 0.5% across all datasets. When network congestion increases to $\geq$ 60%, the adaptive scheme further reduces communication overhead by up to 78% for MNIST and 69% for 5G, while



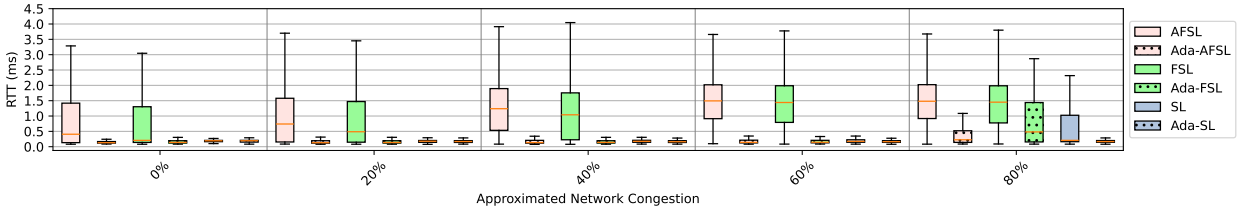
(a) RTT analysis on MNIST dataset.



(b) RTT analysis on 5G dataset.



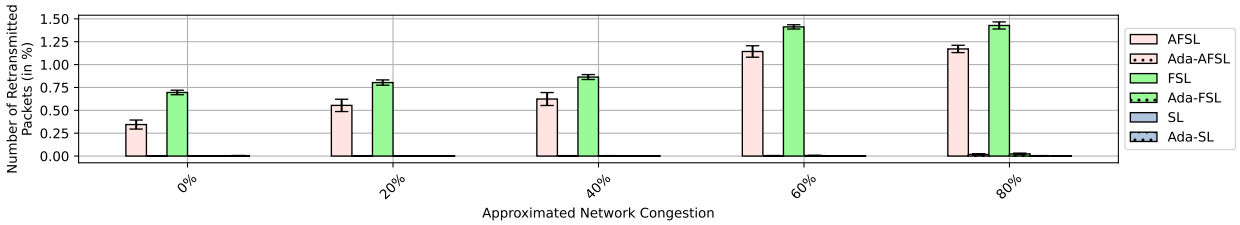
(c) RTT analysis on IoT dataset.



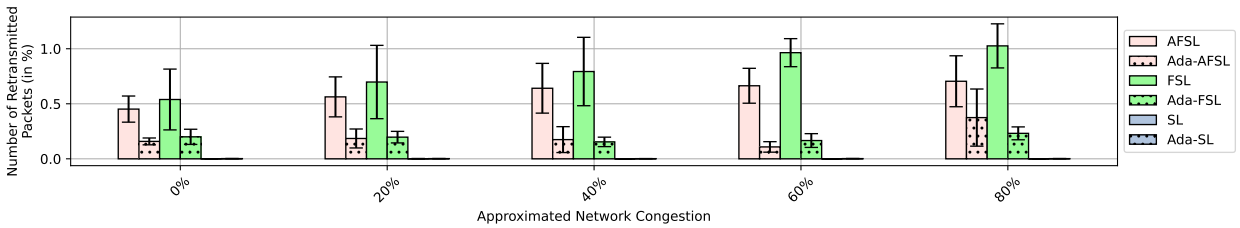
(d) RTT analysis on VoD dataset.

Figure 6.8: Impact on networking KPIs: RTT analysis.

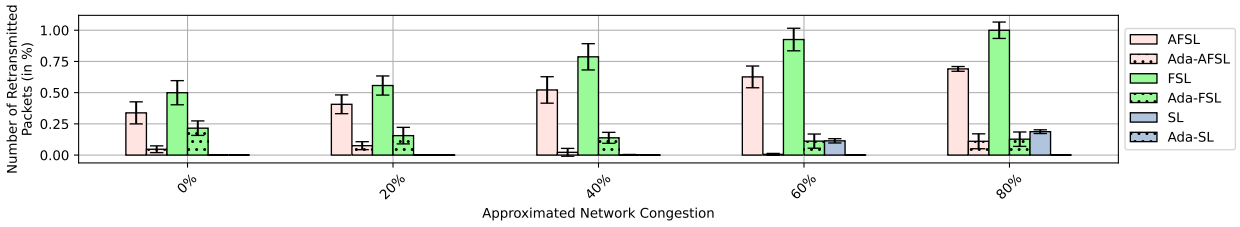
maintaining less than 3% degradation in the F1 score for MNIST. Interestingly, for the 5G dataset, a slight improvement in performance is observed, despite the compression of activations and gradients. This enhancement can be attributed to the ability of compression techniques to prioritize the most relevant features by reducing redundant data, which is especially beneficial for datasets that contain diverse and noisy inputs. The compression techniques effectively serve as an implicit form of regularization, preventing the model from memorizing irrelevant patterns. As a result, the model is better able to generalize by focusing on the key features, ultimately leading to improved performance. It should be noted that performance remained



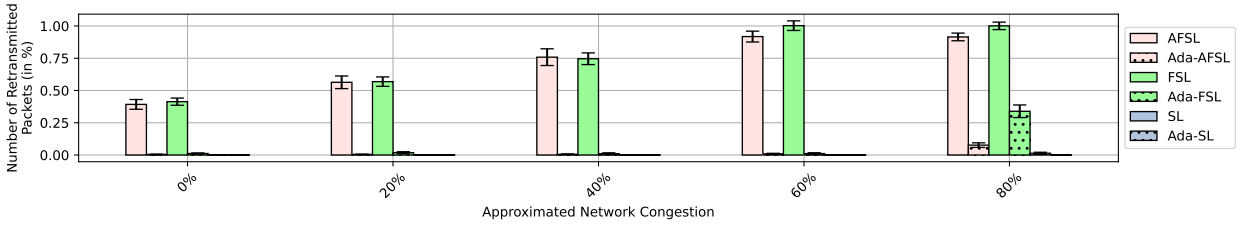
(a) Packet retransmission analysis on MNIST dataset.



(b) Packet retransmission analysis on 5G dataset.



(c) Packet retransmission analysis on IoT dataset.



(d) Packet retransmission analysis on VoD dataset.

Figure 6.9: Impact on networking KPIs: Packet retransmissions.

within the acceptable degradation values (as outlined in Table 6.2). For regressions tasks, in highly congested network, communication overhead is reduced by 81% and 82% for IoT and VoD, respectively. A near zero change in nMean was observed, remaining below the acceptable degradation thresholds. This is achieved by using the $p_{\max}$ values determined in Section 6.5.1, to control the reduction factor and manage the trade-off between performance and communication cost. SL benefits most from the adaptive scheme, as it minimizes performance degradation even in highly congested networks, while achieving communication overhead reductions similar to those of FSL and AFSL. These results demonstrate the robustness and efficiency of the adaptive scheme across different datasets and split learning methods, making it a valuable solution for optimizing communication overhead in dynamic network conditions.

Network Stability Analysis

The network stability of the proposed adaptive scheme is evaluated by assessing key performance indicators (KPIs), including Round Trip Time (RTT), TCP retransmissions and bandwidth utilization. RTT measures the time it takes for a network request to travel from a source to a destination and back to the source. This metric is crucial for assessing connection health and reliability, commonly used by network administrators to diagnose network speed and performance, both in a local network (LAN) and a wide area network (WAN). Packets are discrete units of data transmitted over a network from a source to a destination. In TCP, retransmissions occur when a packet is not acknowledged by the receiver within a specified time frame. This typically happens due to packet loss or corruption in the network, which can result from various factors such as congestion, malfunctioning router, switch configurations, and other issues that impact network stability. Bandwidth utilization refers to the proportion of the available network bandwidth that is being actively used to transmit data. It measures how efficiently the network's bandwidth is being utilized. Efficient bandwidth utilization is crucial for optimizing network performance, especially in scenarios involving large data transfers or high network congestion. In the experiment, the bandwidth utilization U_i of a device i is computed as follows:

$$U_i = \max_{\ell \in \text{PATH}_i^s} (U_\ell), \quad U_\ell = \frac{D_\ell}{\text{CAP}_\ell} \times 100, \tag{6.9}$$

where U_ℓ is the bandwidth utilization of link ℓ, PATH_i^s is the path from device i to server s, D_ℓ the actual data transferred on link ℓ and CAP_ℓ the bandwidth capacity of ℓ.

Figures. 6.8 and 6.9 present the results for RTT and Transmission Control Protocol (TCP) retransmissions obtained with the proposed adaptive scheme, compared to standard approaches. With FSL and AFSL, significant RTT and retransmissions are observed under individual network congestion levels, with both RTT and retransmissions increasing as network congestion rises. In contrast, the adaptive scheme demonstrates a markedly different behavior, with relatively low RTT and retransmissions. The increase in RTT and retransmissions is almost negligible as network congestion increases. Specifically, Ada-SL, Ada-FSL, and Ada-AFSL effectively stabilize RTT up to a critical point (80% network congestion), maintaining a low RTT across all use cases. In highly congested networks ($\geq$ 80%), while Ada-SL maintains a low RTT, Ada-FSL and Ada-AFSL experience a slight increase in RTT. This increase is primarily due to pre-existing network stress, which, despite the adaptive scheme's reduction in traffic, prevents RTT from dropping below the observed values. The RTT stabilization achieved by the proposed approach is crucial for applications such as smart vehicles [Bi et al., 2021] and smart health applications [Gao et al., 2021]. Similar improvements are observed in retransmissions, as shown in Figure 6.9, where a significant reduction in retransmission is achieved, with stable, near-zero retransmission even under high congestion levels. This low retransmission value is particularly beneficial for applications where any retransmission level above 0.5% is unacceptable, such as financial transactions, online gaming, and tele-medicine. In contrast, SL, both with and without the adaptive scheme, exhibits lower RTT and TCP retransmission due to its sequential data transmission, which does not exacerbate congestion, thus minimizing its impact on RTT and TCP retransmission.

As shown in Figure 6.10, where the available bandwidth for the link connecting the device to the server are evaluated, the adaptive scheme (Ada-AFSL) demonstrates a clear advantage as network conditions degrade (indicated by the background color transitioning from green to red). Ada-AFSL consistently maintains higher available capacity with lower RTT compared to standard approaches. In highly congested networks (red background), despite similar available bandwidth, significant differences in RTT are observed. This can be attributed to the adaptive scheme's ability to reduce the volume of data transferred, effectively minimizing the strain on the network. As a result, the network remains less congested, and RTT increases only marginally, demonstrating the scheme's efficiency in maintaining stable network performance even under heavy congestion.

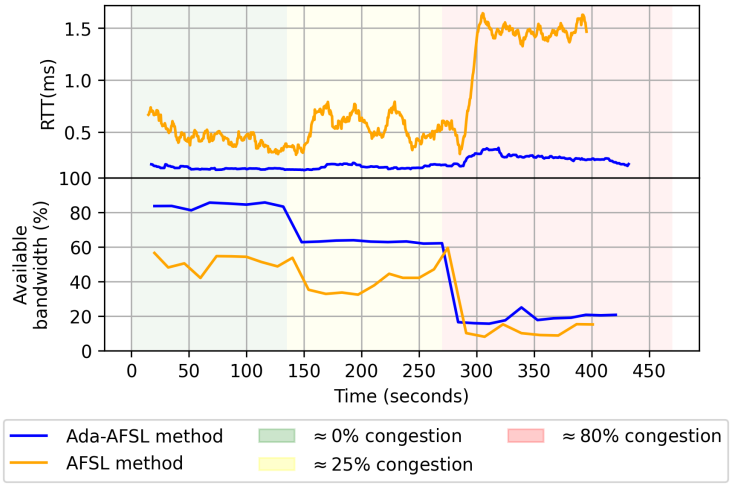


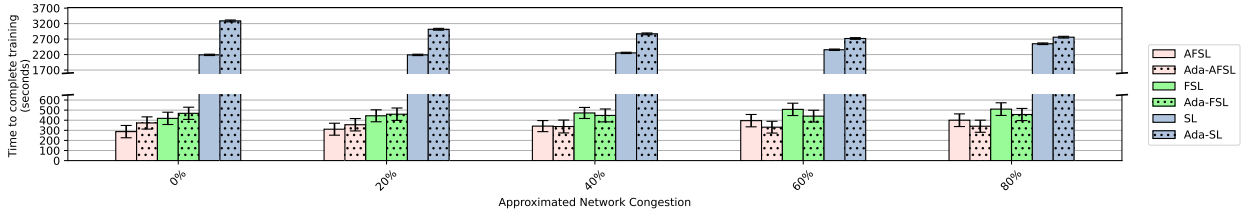
Figure 6.10: Comparison of available bandwidth and RTT between AFSL and Ada-AFSL under varying network congestion levels. By employing dynamic compression, Ada-AFSL effectively reduces bandwidth utilization and RTT.

Training Time Analysis

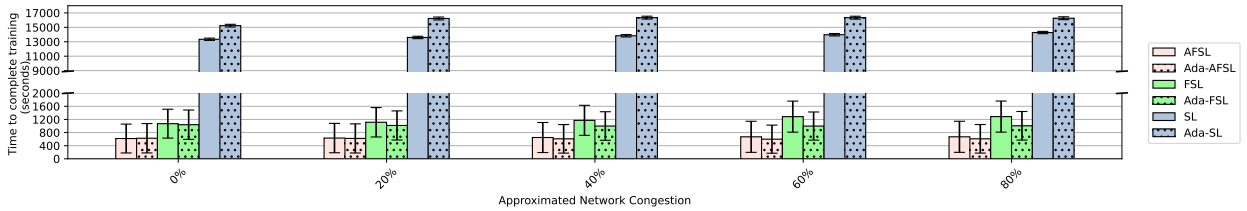
Figure 6.11 presents a comparison of training times across different approaches. Since SL operates sequentially, its training time is significantly higher than that of FSL and AFSL, with AFSL being faster than FSL due to its asynchronous structure. Although the adaptive scheme introduces a compression mechanism on top of the standard approaches, which could initially lead to longer training times, this effect is primarily observed with Ada-SL. For Ada-FSL and Ada-AFSL, the training time gap relative to the standard approaches decreases as network congestion increases.

In low-congestion scenarios ($\leq 20\%$), Ada-FSL and Ada-AFSL indeed require more time to complete a given number of iterations. This includes time for both forward and backward propagation, as well as data transmission across the network. However, as Ada-FSL and Ada-AFSL dynamically adjust and compress data for transmission, as congestion increases (40% to 60% to 80%), the time required for data transfer in Ada-FSL and Ada-AFSL does not increase as significantly as in the standard approaches. This helps accelerate their training processes, ultimately leading to shorter overall training times compared to FSL and AFSL.

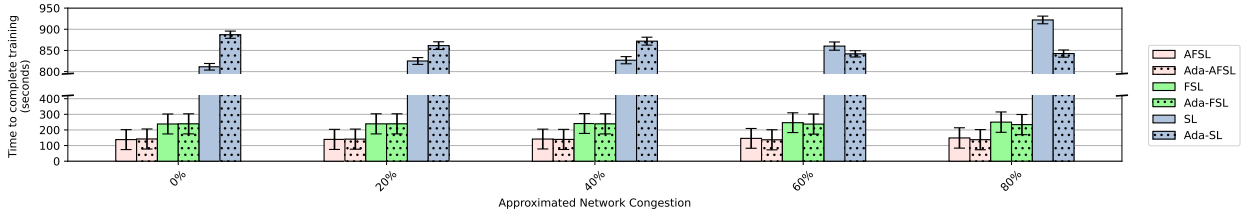
This can be attributed to two factors: first, the time required for data compression remains constant regardless of network conditions. Second, as discussed in Section 6.5.2, the adaptive scheme stabilizes RTT, meaning that while communication time increases with the standard approaches as congestion rises,



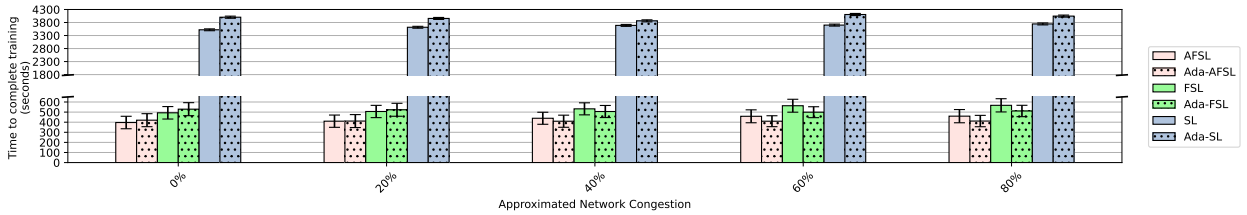
(a) Training time for MNIST dataset.



(b) Training time for 5G dataset.



(c) Training time for IoT dataset.



(d) Training time for VoD dataset.

Figure 6.11: Training time for different datasets.

it remains relatively low with the adaptive scheme. These two factors reduce the time gap between the standard and adaptive approaches, ultimately resulting in a shorter overall training time with the adaptive scheme. SL does not exhibit this behavior because the RTT improvement in Ada-SL over SL is marginal resulting in minimal impact on overall training time. As a sequential process, SL adds the encoder-decoder time per device rather than taking the maximum (as in parallel schemes).

These results underscore the importance of communication costs in distributed model training and suggest that optimizing communication overhead can significantly enhance training efficiency. Further research into this aspect is essential for improving the performance of distributed learning systems.

6.5.3 Training Phase Under Varying Network Stress Profile

In practical scenarios, the network capacity is typically shared with other applications, and its condition may fluctuate over time depending on the network load. To evaluate the effectiveness of the proposed approach under varying network conditions, we assess the impact of the compression scheme on the validation F1score and data transfer during the training phase. For this purpose, without loss of generality, we compare both Ada-AFSL and AFSL methods using the MNIST dataset across three different varying network conditions, referred to as profiles. To simulate network congestion, the *iperf3* tool was used. The stress levels within each profile are detailed in Table 6.5, with the durations of the periods set to 135s, 135s, and 250s, respectively. Note that a network congestion of ≈0% indicates that no other applications are simultaneously using the network, while a congestion of ≈100% means that the network bandwidth is nearly fully utilized.

Table 6.5: Network congestion profiles adopted in the experiments.

	Period1	Period2	Period3
Profile1	≈0%	≈25%	≈80%
Profile2	≈25%	≈80%	≈0%
Profile3	≈80%	≈25%	≈0%

The results are depicted in Figure 6.12. As can be noted, the proposed system adjusts the data being transferred based on the current network congestion level. The compression rate is dynamically adapted, ensuring that the applied compression level varies according to the current network congestion, while respecting the pre-defined reduction threshold (see Figure 6.6(a)). This behavior is evident in the cumulative data transfer curve. For the AFSL scheme, the curve remains linear, except towards the end, when clients complete the training phase and begin disconnecting. The linearity of the curve indicates that the amount of data being transferred remains constant. In contrast, for the Ada-AFSL scheme, the curve's slope varies based on the network congestion level, demonstrating that the data transfer adapts dynamically to network conditions. A total reduction in data transferred of 65.5%, 68.5%, and 68.8% is achieved for Profiles 1, 2, and 3, respectively. It is noteworthy that, although the profiles have identical congestion levels arranged in different sequences, the durations of the periods vary, leading to differences in overall compression. Moreover, during the final period, the timing of client disconnections may vary, affecting the amount of data being transferred. Finally, the drop in the F1 score for the selected model, intended to be used during the inference phase, remains within the predefined threshold of 3%. It is worth mentioning that similar behavior was observed with different SL-based methods and datasets, validating the benefits of the proposed scheme

in reducing data transfer during the training phase.

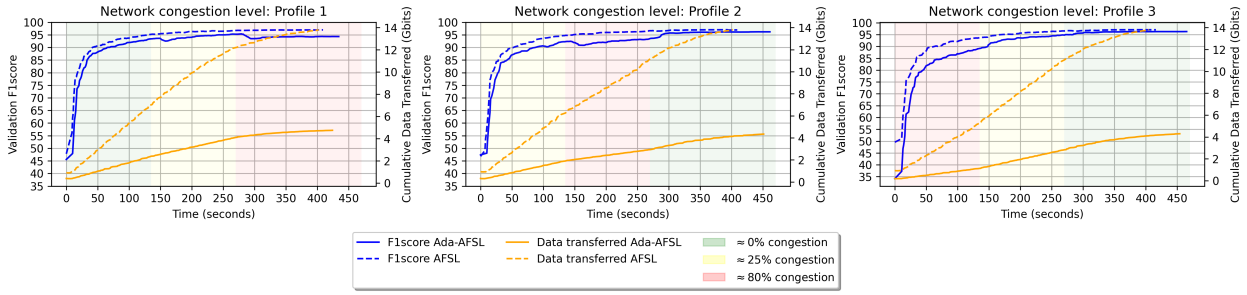


Figure 6.12: Training phase under varying network stress profile. A significant reduction in data transfer is observed, while maintaining the predefined maximum accuracy drop (threshold). The MNIST dataset was used for this experiment.

6.5.4 Inference Phase Under Varying Network Stress Profile

In machine learning, the inference phase is a critical stage where a trained model is used to make predictions on unseen data. In other words, it enables the trained model to work with live data, putting it into production mode. To assess the impact of our proposed approach during the inference phase, three different pre-trained models using the MNIST dataset were utilized: 1) AFSL (baseline) 2) Ada-AFSL trained in a network with 0% congestion and 3) Ada-AFSL trained in a network with 80% congestion. We then evaluated their performance under varying network stress profiles using the MNIST testset as input. To simulate different levels of network stress during the inference phase, the same congestion profiles described in the previous section were applied, as summarized in Table 6.5. The results are presented in Figure 6.13. Similar to the training phase, the proposed scheme effectively reduced the amount of data transferred by

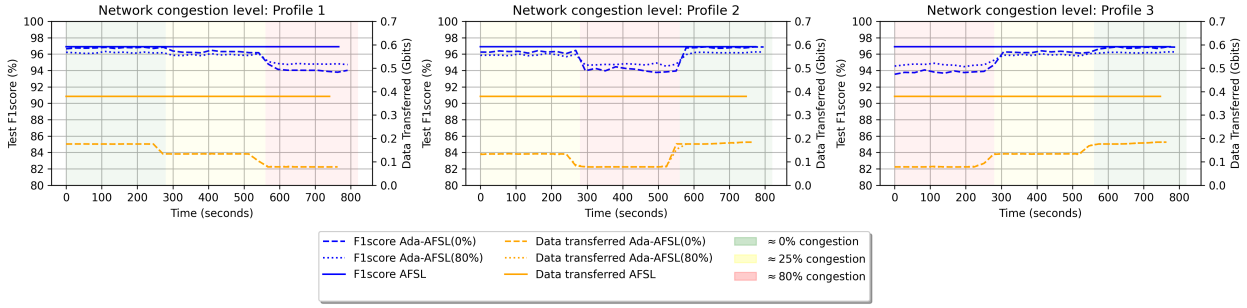


Figure 6.13: Inference phase under varying network stress profiles for three different pre-trained models: 1) AFSL 2) Ada-AFSL trained in a network with 0% congestion and 3) Ada-AFSL trained in a network with 80% congestion. The impact on F1 score and data transferred is presented. The MNIST dataset was used for this experiment.

adjusting the compression scheme based on the current network condition. An average reduction in data transfer of approximately 51%, 61%, and 79% was achieved during periods of ≈0%, ≈25%, and ≈80% network congestion, respectively, for both Ada-AFSL trained models.

Regarding the F1 score, for the Ada-AFSL model trained in a network with ≈0% congestion, the average drop during the inference phase were 0.21%, 0.64% and 2.87% for ≈0%, ≈25% and ≈80% congestion levels, respectively. For the model trained in a network with ≈80% congestion, the observed F1 score drops, during the inference phase, were 0.67%, 1.04% and 2.17% for ≈0%, ≈25% and ≈80% congestion levels, respectively. These findings suggest that the network condition during the training phase influence the model's performance during the inference phase, making the model more specialized for the type of data observed during the training (i.e., more or less compressed data). However, it is important to note that the drop in F1 score remained below the pre-defined threshold of 3% (see Figure 6.6(a)). In summary, the results indicate that a substantial reduction in data transfer can be achieved during the inference phase at the cost of a controlled and pre-defined accuracy degradation.

6.6 Conclusion

In this chapter, we proposed a dynamic compression technique for split learning that adapts to real-time network conditions, significantly reducing communication overhead by up to 82% while preserving model accuracy. Unlike fixed-compression methods, our approach flexibly balances bandwidth usage and performance, with added benefits like improved generalization in some cases. Compatible with various SL architectures and neural network models, this method is well-suited for scalable and efficient deployment in bandwidth-constrained and latency-sensitive environments such as 5G and 6G networks.

Chapter 7

Spatio-temporal Split Learning for Energy-Aware Proactive 5G/B5G Resource Management

7.1 Introduction

As 5G networks scale and evolve toward Beyond 5G (B5G) and 6G architectures, operators face dual challenges: increasing complexity in traffic behavior and heightened demands for energy-efficient, SLA-compliant resource management. The dynamic nature of 5G services, ranging from enhanced Mobile Broadband (eMBB) to Ultra-Reliable Low-Latency Communications (URLLC) and massive Machine-Type Communications (mMTC) places stringent requirements on the availability of compute, memory, and storage resources across distributed sites. These services, implemented through logical slices on shared infrastructure, give rise to diverse and fluctuating traffic patterns that vary across time, space, and service class.

To cope with these fluctuations, accurate traffic forecasting has emerged as a key enabler for proactive resource allocation. By anticipating future demand, operators can scale resources just in time, avoiding the inefficiencies of over-provisioning and the risks of under-provisioning. This is especially important for energy-aware strategies like micro-sleeps and dynamic voltage scaling, which can significantly reduce power consumption, but only when applied with confidence in future load patterns [Al-Karawi et al., 2024; Bahreini et al., 2019].

However, the spatio-temporal behavior of network slices is both heterogeneous and localized. Business districts experience daytime eMBB peaks; residential zones see evening traffic surges; industrial IoT flows may follow highly periodic schedules. Notably, some geographically distant sites can exhibit remarkably similar traffic behaviors due to socio-economic or functional similarities. These latent correlations are not necessarily reflected by physical distance or raw statistical similarity, making it difficult to model inter-node relationships using standard approaches based on distance metrics, Pearson correlation, or local Moran's I [Zhao et al., 2021a; Mortazavi and Sousa, 2023].

To address these challenges, we propose a two-stage solution. First, we introduce ST-SplitGNN, a decentralized spatio-temporal forecasting model based on split learning. It allows each site to learn its own temporal traffic patterns in a privacy-preserving manner while sending intermediate representations to a central server. There, a Graph Neural Network (GNN) is used to uncover non-obvious inter-site dependencies via a learnable adjacency matrix, improving global forecasting performance.

Building on this, we present ST-SplitGNN+, a hierarchical, uncertainty-aware resource management framework. Unlike prior work that relies solely on point forecasts [Xu et al., 2022; Amiri and Mohammad-Khanli, 2017], ST-SplitGNN+ uses both point predictions and uncertainty intervals to guide resource allocation decisions. By quantifying the confidence around traffic forecasts, our model enables fine-grained, proactive provisioning of CPU, memory, and storage resources, ensuring SLA compliance while avoiding energy waste. For instance, during periods of high certainty, resources can be aggressively downsized to save energy; during uncertain peaks, conservative scaling mitigates performance risk.

Our solution addresses key limitations in prior work: traditional forecasting models fail to model complex spatial relationships; centralized approaches compromise privacy and scalability; and resource allocation schemes often ignore uncertainty, leading to brittle or wasteful provisioning [Kabir et al., 2021; Kasuluru et al., 2024; Wu et al., 2021]. In contrast, ST-SplitGNN+ combines spatio-temporal split learning, learned graph structures, and uncertainty-aware reasoning to jointly optimize prediction quality and resource allocation effectiveness in large-scale, decentralized 5G/B5G networks composed of Radio Units (RUs), Distributed Units (DUs), Centralized Units (CUs), and User Plane Functions (UPFs).

7.2 Related Work

With the increasing complexity and scale of 5G and Beyond 5G (B5G) networks, intelligent traffic prediction and resource management have become vital for ensuring service reliability, energy efficiency, and compliance with service-level agreements (SLAs). Numerous approaches have been proposed to tackle these challenges, particularly using machine learning (ML) and deep learning (DL) paradigms. However, most existing works suffer from one or more of the following limitations: lack of privacy preservation, static spatial modeling, limited incorporation of uncertainty, or separation between forecasting and resource provisioning tasks. In what follows, we review the state of the art in traffic forecasting, split learning, GNN-based modeling, and uncertainty-aware resource allocation to position our contribution.

7.2.1 Spatio-Temporal Traffic Forecasting in 5G Networks

Traffic forecasting is a well-studied topic, especially in the context of intelligent transportation and communication networks. Early deep learning approaches leveraged recurrent neural networks (RNNs), such as LSTMs and GRUs, to model temporal dependencies. However, these models failed to capture spatial interactions between network nodes. To address this, spatio-temporal graph neural networks (ST-GNNs) have been introduced, combining graph convolutional networks (GCNs) with recurrent layers. For instance, the STGCN model [Yu et al., 2017] introduced a framework for joint spatial and temporal learning using fixed graph structures. Similarly, T-GCN [Zhao et al., 2019] fused GCNs and GRUs for traffic prediction on road networks. Futhermore, TSGAN [Wang et al., 2022b] combines dynamic time warping (DTW) and graph attention networks, which can extract and utilize the spatial-temporal dependencies and characteristics of cellular traffic data and consider the differentiated influence between adjacent cells and core cells. More recent work has explored adaptive graph learning, where inter-node relationships are learned during training instead of being pre-defined. STAGCN [Ma et al., 2023] and ADSTGCN [Cui et al., 2023] and more [Xiong et al., 2024; An et al., 2025] demonstrate improved performance through dynamic adjacency matrices. However, these models are largely designed for transportation systems and operate in centralized architectures, making them unsuitable for privacy-constrained edge network environments in 5G/B5G.

7.2.2 Split Learning and Privacy-Preserving Forecasting

In scenarios where raw traffic data cannot be shared due to privacy, regulatory, or ownership concerns, split learning (SL) has emerged as a viable alternative. SL divides the learning task between edge devices and a central server, with clients training partial models locally and only exchanging intermediate representations. Lin *et al.* [Lin et al., 2024a] proposed a SL architecture tailored for 6G edge networks, aiming to reduce data exposure and computation load. Iqbal *et al.* [Iqbal et al., 2025] proposed a privacy-preserving collaborative split learning framework for smart grid load forecasting, demonstrating the effectiveness of SL in sensitive, distributed environments where raw data sharing is not feasible. Further work on split federated learning [Hafi et al., 2024; Albuquerque et al., 2024] merges SL with federated learning concepts to balance scalability and privacy. However, these models do not account for spatial dependencies between edge sites or incorporate temporal forecasting capabilities, limiting their applicability to dynamic traffic prediction. Furthermore, SL has not yet been effectively integrated with graph learning or uncertainty quantification, which are essential for 5G/B5G traffic modeling and proactive network management.

7.2.3 Graph Neural Networks for Modeling Inter-Node Dependencies

Graph Neural Networks (GNNs) have demonstrated strong capabilities in modeling the complex relational structures inherent in distributed communication networks. For example, Eisen *et al.* [Eisen et al., 2019] proposed GNNs with randomized edge connectivity to optimize wireless scheduling tasks efficiently. Other works, such as RouteNet-Fermi[Ferriol-Galmés et al., 2023b] and its variants [Ziazet et al., 2022b], further showcase the power of GNNs in capturing the intricate dependencies between network topology, routing policies, and traffic demands. These models accurately estimate per-flow metrics such as delay, jitter, and packet loss, highlighting the effectiveness of GNNs in understanding inter-node relationships. This advantage is also emphasized in [Song et al., 2024], where graph attention networks and transformer-based architectures are used to model vehicular networks. These models enable a nuanced understanding of both the network topology and its evolving usage patterns.

Despite their strengths, most GNN-based solutions assume full data visibility at a centralized controller and often ignore privacy-preserving mechanisms like SL. Additionally, existing approaches rarely account for the uncertainty in predicted workloads, which is crucial for robust decision-making in dynamic and volatile traffic conditions.

7.2.4 Uncertainty-Aware Resource Management

Proactive resource allocation guided by traffic prediction is essential for improving energy efficiency and quality of service (QoS) in modern networks. However, when predictive uncertainty is ignored or underestimated, resource provisioning decisions can become suboptimal, resulting in either under-provisioning (leading to SLA violations) or over-provisioning (causing unnecessary energy consumption). In [Bahreini et al., 2019], virtual machine allocation based on point forecasts was shown to be energy-efficient, but the approach lacked mechanisms to account for forecast uncertainty. Kabir *et al.* [Kabir et al., 2021] emphasized the importance of uncertainty quantification in edge resource orchestration, demonstrating that probabilistic forecasting can significantly reduce the risk of SLA violations. More recently, probabilistic forecasting techniques have been applied to both terrestrial and non-terrestrial network scenarios [Kasuluru et al., 2023, 2024; Vaca-Rubio et al., 2025]. These works explore methods such as simple multi-layer perceptrons, DeepAR [Flunkert et al., 2017], and Transformer-based models [Vaswani et al., 2017], which estimate the parameters of an assumed Gaussian distribution by maximizing the Gaussian log-likelihood. When evaluated for their ability to characterize physical resource block (PRB) load dynamics, these models demonstrate strong potential in providing accurate forecasts while quantifying predictive uncertainty.

Although these studies underscore the utility of probabilistic forecasting and prediction intervals in resource management, they do not incorporate spatial dependencies across network nodes, nor do they support decentralized or privacy-preserving learning paradigms. As such, they fall short of addressing the unique challenges posed by 5G/B5G edge networks, where distributed learning, data sensitivity, and dynamic spatio-temporal behavior must be simultaneously considered.

7.2.5 Our Contributions

Based on the reviewed literature, several critical research gaps remain unaddressed:

- **Privacy limitations and lack of model specialization:** Most deep learning-based traffic prediction frameworks assume centralized access to raw data, which poses significant privacy and data-sharing concerns in distributed 5G/B5G environments. Moreover, centralized models often fail to capture the localized and site-specific characteristics of traffic patterns, leading to suboptimal predictive performance.

- **Static or predefined spatial modeling:** Many GNN-based approaches rely on fixed or manually

defined adjacency matrices, which limits their ability to capture dynamic and latent inter-site relationships that evolve over time or are not directly linked to physical proximity.

- **Lack of uncertainty quantification and integrated frameworks:** Most existing forecasting methods provide only point estimates and lack prediction intervals, reducing their robustness in resource provisioning under fluctuating traffic demands. While some methods incorporate uncertainty, they often overlook spatial dependencies and do not support privacy-preserving, decentralized learning, making them impractical for real-world 5G/B5G deployments.

To address these gaps, we propose ST-SplitGNN+, a novel two-stage framework that unifies decentralized spatio-temporal learning with uncertainty-aware resource management:

(1) **ST-SplitGNN**: A decentralized, privacy-preserving split learning model in which each site independently learns the temporal characteristics of its local traffic (comprising multiple slices) using a site-specific encoder. The encoder generates activations that are transmitted rather than raw data to a central server, reducing privacy risks. There, a Graph Neural Network (GNN) with a learnable adjacency matrix captures latent spatial inter-site dependencies. Unlike fixed graphs based on geographic distance or static network topology, the learnable adjacency matrix enables the GNN to dynamically infer and adapt to evolving inter-site correlations, including those driven by demand patterns, routing policies, or shared content interests, capturing relationships that are not evident from static metrics alone.

(2) **ST-SplitGNN+**: An extension of the base forecasting model that incorporates uncertainty quantification through both point and quantile predictions. While point forecasts provide expected traffic volumes, quantile predictions capture the full range of potential demand fluctuations, enabling the system to assess and prepare for worst-case and best-case scenarios. This richer uncertainty information helps avoid both under-provisioning (which can lead to SLA violations) and over-provisioning (which wastes energy and resources). These predictions feed into a hierarchical, site-level resource allocation mechanism (following the RU-DU-CU-UPF architecture), that proactively provisions CPU, memory, and storage resources based on both demand forecasts and associated uncertainty. This enables fine-grained scaling decisions that minimize SLA violations and improve energy efficiency.

In summary, our framework jointly addresses four key challenges in next-generation network management: (i) accurate per-site, per-slice spatio-temporal traffic prediction, (ii) decentralized and privacy-aware learning, (iii) adaptive spatial relationship modeling, and (iv) uncertainty-aware, proactive resource provisioning. To the best of our knowledge, this is the first integrated solution tailored for large-scale, dynamic, and distributed 5G/B5G network environments.

7.3 Proposed Uncertainty-aware Spatio-temporal Split GNN Framework

The proposed solution follows a two-stage architecture consisting of a *predictive head* and a *resource allocation head* (see Figure 7.1). The first stage, termed ST-SplitGNN, is responsible for traffic forecasting across multi-site, multi-slice 5G/B5G network architecture composed of Radio Units (RUs), Distributed Units (DUs), Centralized Units (CUs), and User Plane Functions (UPFs). It leverages a spatio-temporal split learning framework to jointly capture localized temporal dynamics and inter-site spatial dependencies. Each client (e.g., Radio Unit) locally trains a temporal model based on its own traffic slice patterns, while the server aggregates intermediate representations using a graph neural network (GNN) with a learnable adjacency matrix. Building upon the forecasts and associated uncertainty quantification from ST-SplitGNN, the second stage, ST-SplitGNN+, acts as a hierarchical resource allocation module. It uses both point estimates and prediction intervals to proactively assign compute, memory, and storage resources per site. This uncertainty-aware allocation process enables fine-grained, adaptive provisioning that balances energy efficiency with service-level agreement (SLA) compliance. In the following sections, we first detail the design and training process of the ST-SplitGNN prediction model, followed by the architecture and decision logic underpinning the ST-SplitGNN+ resource allocation strategy.

7.3.1 System Model

We consider a hierarchical 5G/B5G network architecture composed of RUs, DUs, CUs, and UPFs, represented by the node sets $V_{\text{RU}}, V_{\text{DU}}, V_{\text{CU}}, V_{\text{UPF}}$, respectively. As illustrated in Figure 7.1, these components are organized in a tree-like topology: each RU is connected to a DU, each DU to a CU, and each CU to one or more UPFs [O-RAN Alliance, 2022]. This hierarchical connectivity reflects the typical functional split in 5G disaggregated architectures. The mapping between RUs, DUs, CUs, and UPFs is assumed to be known from the placement strategy [Ziazet et al., 2023], along with the physical location of each node. Each node

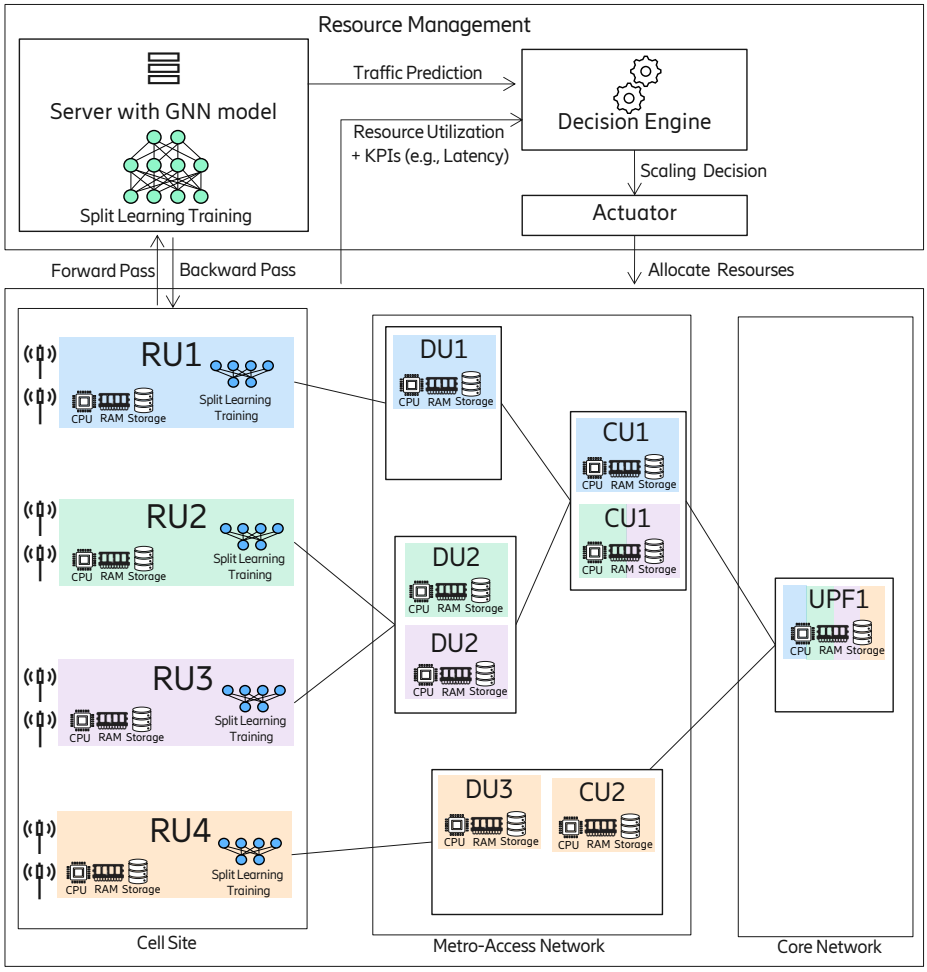


Figure 7.1: System overview of ST-SplitGNN+, the proposed spatio-temporal split learning framework for resource allocation.

in this architecture runs its respective function on a logical *processing instance*, which can be instantiated on virtual machines (VMs), or containers depending on deployment constraints and operator policy. We assume that each processing instance can be scaled in real time with minimal delay, either vertically, by adjusting allocated computing resources (such as CPU and memory), or horizontally, by activating or deactivating individual instances on demand.

The network is modeled as a directed graph $G = (\mathcal{V}, E)$, where: $V = V_{\text{RU}} \cup V_{\text{DU}} \cup V_{\text{CU}} \cup V_{\text{UPF}}, E \subseteq V \times V$, and the edges E represent the logical communication links in the node hierarchy. The goal of the system is to dynamically allocate compute (CPU), memory, and storage resources across all nodes. The objectives are twofold: (i) to minimize energy consumption across the network, and (ii) to satisfy strict Quality of Service (QoS) requirements, particularly end-to-end latency constraints.

7.3.2 Prediction Head: A Spatio-temporal Split GNN Model

The first stage of the proposed solution is to predict the traffic at each RU for each slice that will later be used to infer the traffic at DU, CU and UPF levels for the resource allocation task. We consider a 5G/B5G network composed of N RU sites ($N = |V_{RU}|$), each responsible for traffic management on S distinct network slices (e.g., eMBB, URLLC, mMTC). Let $x_{v,s}^{(t)} \in \mathbb{R}$ denote the traffic volume at node $v \in V_{\text{RU}}$ and slice $s \in \{1, \ldots, S\}$ at time t. Let $\mathbf{X}_v^{(t)} = [x_{v,1}^{(t)}, x_{v,2}^{(t)}, \ldots, x_{v,S}^{(t)}]^\top \in \mathbb{R}^S$ denote the slice-wise traffic vector at node v at time t. Over a time window of length T, the temporal input traffic history at node v is denoted as:

$$\mathbf{I}_v^{(t)} = [\mathbf{X}_v^{(t-T+1)}, \mathbf{X}_v^{(t-T+2)}, \ldots, \mathbf{X}_v^{(t)}] \in \mathbb{R}^{T \times S}. \tag{7.1}$$

Given that historical traffic $\mathbf{I}_v^{(t)}$, the goal here is to predict the traffic volumes over the next Q time steps for all nodes and all slices:

$$\hat{\mathbf{I}}_v^{(t)} = [\hat{\mathbf{X}}_v^{(t+1)}, \hat{\mathbf{X}}_v^{(t+2)}, \ldots, \hat{\mathbf{X}}_v^{(t+Q)}] \in \mathbb{R}^{Q \times S}. \tag{7.2}$$

This task is inherently a spatio-temporal multivariate time series forecasting problem, as it requires learning both:

- **Temporal dynamics**: periodic trends, bursty behaviors, and slice-specific activities at each node.
- **Spatial dependencies per RU at a given time**: correlations and influences across different network nodes, which may vary in time and are not necessarily based on geographical proximity.

The key challenge lies in jointly modeling these heterogeneous spatio-temporal patterns to produce accurate and scalable multi-step predictions.

Architecture of ST-SplitGNN

As shown in Figure 7.2, the proposed prediction head (i.e., ST-SplitGNN) is composed of two main components: *(i)* client-side local temporal encoders that specialize in modeling slice-wise traffic dynamics at each RU, and *(ii)* a server-side graph neural network that aggregates node embeddings while learning inter-node dependencies via a learnable adjacency matrix. This framework allows each node to specialize in its traffic pattern while allowing the server to model dependencies between RU nodes in an adaptive and task-oriented manner.

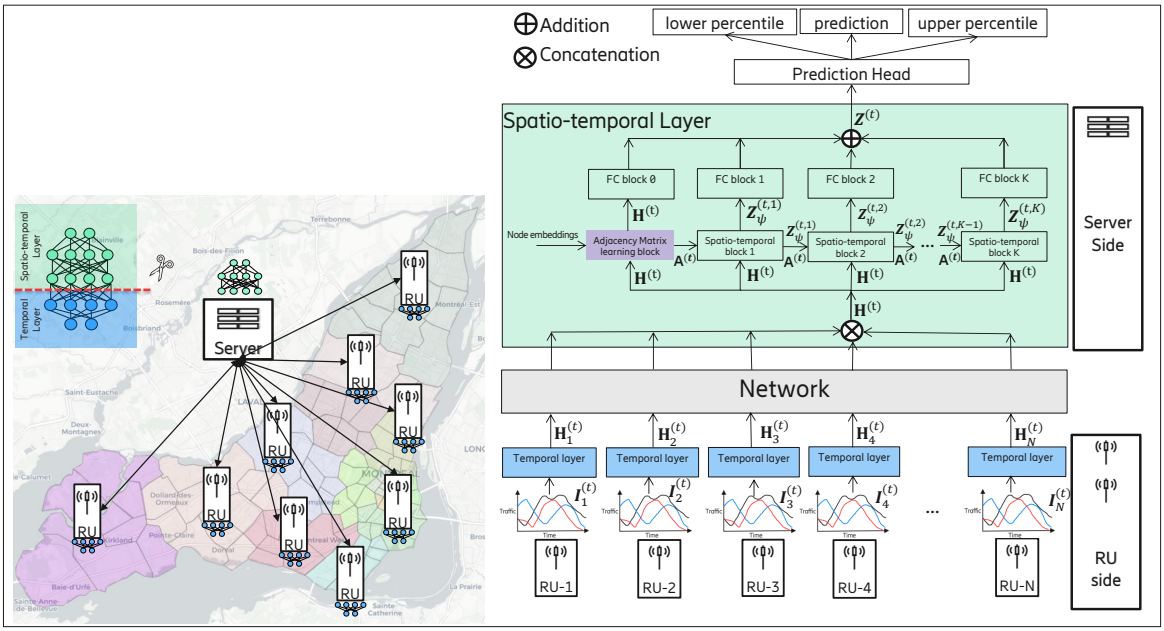


Figure 7.2: System overview of ST-SplitGNN, the proposed spatio-temporal split learning framework. Each node processes its local slice-wise traffic data using a temporal encoder. Encoded embeddings are sent to a central server, which aggregates them via a GNN with a learnable adjacency matrix to predict future traffic.

Client-Side Temporal Encoder

Each RU node v maintains a local model that processes its own slice-wise input traffic history $\mathbf{I}_v^{(t)} \in \mathbb{R}^{T\times S}$, where T is the time window size and S is the number of slices. The local encoder $f_{\theta_v}(\cdot)$ is implemented as a temporal model (e.g., LSTM, GRU, or Transformer) and learns to extract a compact latent representation of the node's recent traffic behavior:

$$\mathbf{H}_v^{(t)} = f_{\theta_v}(\mathbf{I}_v^{(t)}) \qquad \mathbf{H}_v^{(t)} \in \mathbb{R}^{D\times S}, \quad v \in V_{\text{RU}}, \tag{7.3}$$

where $\mathbf{H}_v^{(t)}$ is the local embedding (activations) of node v at time t, D the embedding dimension, and θ_v the model parameters, i.e., weights and biases of RU node v. This representation is then transmitted to the server while preserving privacy (no exposure of raw traffic data), and reducing communication overhead.

Server-Side Graph Aggregation with Learnable Adjacency

The central server collects embeddings $\mathbf{H}^{(t)}$, ($\mathbf{H}^{(t)} = [\mathbf{H}_1^{(t)}, \ldots, \mathbf{H}_N^{(t)}]^\top \in \mathbb{R}^{N\times D\times S}$) from all N RU nodes and aggregates them using a Graph Neural Network (GNN). The server learns a task-driven

adjacency matrix $\mathbf{A}^{(t)} \in \mathbb{R}^{N\times N}$ that captures spatial dependencies between the RU nodes. Unlike fixed adjacency matrices based on distance or correlation, $\mathbf{A}^{(t)}$ is optimized during training to improve predictive accuracy. The graph aggregation process is defined as:

$$\mathbf{Z}^{(t)} = \text{GNN}_{\psi}(\mathbf{H}^{(t)}, \mathbf{A}^{(t)}), \tag{7.4}$$

where $\mathbf{Z}^{(t)} = [\mathbf{Z}_1^{(t)}, \ldots, \mathbf{Z}_N^{(t)}]^\top \in \mathbb{R}^{N\times\ D\ \times\ S}$ is the GNN-aggregated representation and ψ is the server side model parameters, i.e., weights and biases. The GNN can be instantiated using any modified architecture of GNN, e.g., GCN, GAT, or ST-GCN layer, where the adjacency matrix is a learnable parameter instead of a fixed input.

Prediction Head and Loss Function

For each RU node v, the server uses a prediction head $g_\phi(\cdot)$ (e.g., a multilayer perceptron) to forecast the traffic demand over the next Q time steps, including both a point estimate and uncertainty bounds in the form of quantile predictions. The output is:

$$\hat{\mathbf{I}}_v^{(t)} = g_\phi(\mathbf{Z}_v^{(t)}), \tag{7.5}$$

where $\hat{\mathbf{I}}_v^{(t)} = \left[\hat{\mathbf{X}}_v^{(t+1)}, \ldots, \hat{\mathbf{X}}_v^{(t+Q)}\right] \in \mathbb{R}^{Q\times S\times R}$ denotes the multi-step forecasts for S slices and R outputs per prediction (e.g., lower quantile, point estimate, upper quantile). The input $\mathbf{Z}_v^{(t)}$ is the GNN-aggregated spatio-temporal representation of RU node v at time t.

Instead of using only point estimates with standard loss functions like mean squared error, we adopt quantile regression to enable uncertainty-aware forecasting. Specifically, we are optimizing the asymmetric least absolute deviations error, better known as *pinball loss* function Steinwart and Christmann [2011], which allows the model to learn multiple quantiles $\{\tau_1, \tau_2, \ldots, \tau_R\}$ (e.g., $\tau = 0.1, 0.5, 0.9$) of the conditional distribution of the traffic. The total loss is computed as:

$$\mathcal{L}_{\text{pinball}} = \frac{1}{NQSR} \sum_{i=1}^{N} \sum_{q=1}^{Q} \sum_{s=1}^{S} \sum_{r=1}^{R} \rho_{\tau_r} \left(y_{i,q,s} - \hat{y}_{i,q,s}^{(\tau_r)} \right), \tag{7.6}$$

where $y_{i,q,s}$ is the ground-truth traffic of the s-th slice at time step q of the i-th sample, $\hat{y}_{i,q,s}^{(\tau_r)}$ is the predicted quantile τ_r, and ρ_τ is the pinball loss function defined as:

$$\rho_\tau(u) = \begin{cases} \tau u, & \text{if } u \geq 0 \\ (\tau - 1)u, & \text{if } u < 0 \end{cases}. \tag{7.7}$$

This loss asymmetrically penalizes over- and under-estimations, making it suitable for learning quantile boundaries. In our model, $\tau = 0.5$ corresponds to the median (point forecast), while $\tau = 0.1$ and $\tau = 0.9$ are used to estimate the lower and upper bounds of the prediction interval. By optimizing this loss function, the prediction head learns not only to forecast traffic volume but also to quantify its uncertainty, an essential feature for downstream resource allocation in ST-SplitGNN+.

Design of the GNN Model

The GNN model, denoted as GNN_ψ, is included to capture the spatial dependencies between RU nodes by propagating and aggregating information across a learned graph structure. This design accounts for the fact that correlations between nodes are not static but may vary over time. Due to changing user activity and traffic patterns, certain nodes may exhibit strong correlations at one moment and weak or no correlations at another. Our model aims to capture this dynamic nature of node correlations.

Standard GNNs face two major limitations: *(i)* **Over-smoothing**: As message passing layers increase, node representations tend to converge and lose their discriminative power [Li et al., 2018a]; and *(ii)* **Noisy propagation**: If spatial dependencies are weak or nonexistent, aggregating neighbor features can introduce noise instead of useful context. To address these issues, we adopt a residual propagation strategy inspired by [Wu et al., 2020b], which balances the original node features with neighbor aggregation. Specifically, we allow each node to retain part of its initial representation at each GNN layer. The layer-wise update at depth k and time t is defined as:

$$\mathbf{Z}_\psi^{(t,k)} = \alpha \mathbf{H}^{(t)} + (1 - \alpha)\tilde{\mathbf{A}}^{(t)} \mathbf{Z}_\psi^{(t,k-1)}, \tag{7.8}$$

where:

- $\mathbf{H}^{(t)}$ is the input embedding at time t (i.e., output from the temporal module),
- $\mathbf{Z}_\psi^{(t,k)}$ is the output of the k-th GNN layer ($\mathbf{Z}_\psi^{(t,0)} = \mathbf{H}^{(t)}$),
- $\alpha \in [0, 1]$ is a residual weighting hyperparameter,

- $\tilde{\mathbf{A}}^{(t)}$ is the symmetrically normalized adjacency matrix generated at time t:

$$\tilde{\mathbf{A}}^{(t)} = \tilde{\mathbf{D}}^{(t)^{-1}}(\mathbf{A}^{(t)} + \mathbf{I}),$$

where $\tilde{\mathbf{D}}_{ii}^{(t)} = 1 + \sum_j \mathbf{A}_{ij}^{(t)}$ ensures normalization with added self-loops.

Learning the Adjacency Matrix: Unlike predefined graphs based on physical topology or correlation metrics, we learn the adjacency matrix $\mathbf{A}^{(t)} \in \mathbb{R}^{N \times N}$ directly from the data. This allows the model to dynamically discover latent spatial dependencies that may arise from similar traffic patterns, even between geographically distant sites. Furthermore, this allows us to have a dynamic adjacency matrix that can change over time capturing the dynamic nature of node correlation. Following the approach in [Wu et al., 2020b], the adjacency matrix is learned using two trainable node embedding matrices, $\mathbf{E}_1$ and $\mathbf{E}_2$. These embeddings are first transformed through learnable weight matrices and nonlinear activations to produce intermediate representations:

$$\bar{\mathbf{E}}_1 = \tanh(\beta \mathbf{E}_1 \Theta_1), \quad \bar{\mathbf{E}}_2 = \tanh(\beta \mathbf{E}_2 \Theta_2), \tag{7.9}$$

where Θ_1 and Θ_2 are learnable parameters, and β is a hyperparameter that controls the saturation rate of the activation function. The final adjacency matrix $\mathbf{A}$ is computed as:

$$\mathbf{A}^{(t)} = \text{ReLU}\left(\tanh\left(\beta(\bar{\mathbf{E}}_1 \bar{\mathbf{E}}_2^\top - \bar{\mathbf{E}}_2 \bar{\mathbf{E}}_1^\top)\right) + \phi(\mathbf{H}^{(t)})\right), \tag{7.10}$$

where $\phi(\mathbf{H}^{(t)})$ learns dynamic temporal influence based on the node features (output from the temporal module) at time t. $\phi_{i,j}(\mathbf{H}^{(t)}) = \phi(\mathbf{H}_i^{(t)} \bigotimes \mathbf{H}_j^{(t)})$ computes pairwise scores using a shared MLP applied to $\mathbf{H}_i^{(t)} \bigotimes \mathbf{H}_j^{(t)}$, the concatenated pairs of node features (activations). This formulation enables the model to learn an adaptive and potentially asymmetric adjacency structure that captures latent dependencies between nodes in a fully data-driven manner. The asymmetric correlation between RUs arises from directional user mobility patterns and non-uniform user demand over time. These factors create one-way dependencies, where traffic at one RU informs or predicts another, but not equally in reverse, especially during periods of peak movement or shifting usage hotspots. For example, at specific times of the day such as morning commutes, lunch breaks, or evening rush hours, users tend to move in dominant directions (e.g., from

residential to commercial areas in the morning, and vice versa in the evening). This introduces a non-reciprocal influence between RUs: the traffic pattern at a source RU (where users are leaving) becomes predictive of the traffic at a target RU (where users are arriving), but not necessarily the other way around. Additionally, variations in user demand (e.g., more streaming or data usage in certain zones) may cause one RU's traffic state to influence another RU's future load, especially if users frequently transition between them. However, this influence might not be symmetric because the volume and behavior of users moving in one direction differ from the reverse direction. Our framework is designed to capture such a dynamic correlation.

To further improve robustness, we apply a set of learnable transformation matrices $\mathbf{W}^{(k)}$ at each layer to filter the propagated information. The final GNN output at time t is aggregated over K layers as:

$$\mathbf{Z}^{(t)} = \sum_{k=0}^{K} \mathbf{Z}_{\psi}^{(t,k)} \mathbf{W}^{(k)}. \tag{7.11}$$

Two desirable properties are therefore provided:

- When spatial correlation is high, $\mathbf{W}^{(k)}$ amplifies neighbor contributions across layers.

- When spatial correlation is weak, the model can suppress inter-node influence by learning $\mathbf{W}^{(k)} \approx \mathbf{0}$ for $k > 0$, preserving only local node representations.

This flexible design enables the GNN to learn both strong and weak spatial interactions in a data-driven manner, while avoiding over-smoothing and noisy propagation.

Advantages of the Split Learning Framework: Our design enables node-level specialization by allowing each client to train on its own traffic history. This is particularly important in heterogeneous 5G environments where traffic patterns vary significantly from location to location. Meanwhile, the centralized GNN dynamically learns latent spatial correlations, improving global coordination for accurate predictions. By decoupling temporal and spatial learning, the proposed method offers scalability, specialization, and efficiency.

7.3.3 Resource Allocation Head: A Greedy Algorithm

Our framework ST-SplitGNN+ leverages traffic prediction with quantified uncertainty to guide vertical (adjusting resources of active nodes) and horizontal scaling (activating or deactivating new instances

decisions, ensuring energy efficiency and SLA compliance. The proposed solution is designed to react to dynamic workloads while maintaining stateful and continuous service delivery.

Given the predicted traffic demands from ST-SplitGNN model introduced in Section 7.3.2, which provides a probabilistic forecast for each RU $v \in V_{\text{RU}}$, the model specifically outputs:

- $\hat{\mathbf{I}}_v^{(t)}$: the point prediction (mean or median), containing the traffic predictions $\hat{x}_{v,s}^{(t)}$ for slice s at time t over a time windows of length Q,
- $\hat{\mathbf{I}}_v^{(t)^{(10)}}$ and $\hat{\mathbf{I}}_v^{(t)^{(90)}}$: the predicted 10th percentile and 90th percentile respectively representing the lower and upper bounds of the prediction interval.

This prediction interval captures uncertainty in future traffic and enables the controller to make risk-aware scaling decisions. While the 10th and 90th percentiles are used in this study, the framework can be extended to support arbitrary percentiles for finer-grained control.

Traffic flows upward from RUs to the core or downwards from core to RU. Each node aggregates the predicted traffic from its children in the hierarchy. Thus, for any node $v \in V \setminus V_{\text{RU}}$, the aggregated traffic for the slice s at time t is given recursively by:

$$\hat{x}_{v,s}^{(t)} = \sum_{u \in \mathcal{C}(v)} \hat{x}_{u,s}^{(t)}, \tag{7.12}$$

where $\mathcal{C}(v)$ denotes the set of child nodes of v in the graph G. This hierarchical aggregation reflects the typical traffic flow from end devices through RUs towards the UPFs, enabling consistent scaling decisions at each layer.

Resource Model

Each node $v \in V$ in the network requires a certain amount of resources, CPU, memory, and storage, to process its incoming traffic. The traffic load at node v at time t for slice s is denoted by $x_{v,s}^{(t)}$, and the corresponding allocated resource vector is represented as:

$$R_v^{(t)} = \left[\text{CPU}_v^{(t)}, \text{RAM}_v^{(t)}, \text{STO}_v^{(t)} \right],$$

where $\text{CPU}_v^{(t)}$ is the required CPU, $\text{RAM}_v^{(t)}$, the required memory, and $\text{STO}_v^{(t)}$ the required storage.

We assume linear relationships between traffic and resource consumption:

$$\text{CPU}_v^{(t)} = \kappa_v^{\text{CPU}} \sum_{s=1}^{S} x_{v,s}^{(t)}, \tag{7.13}$$

$$\text{RAM}_v^{(t)} = \kappa_v^{\text{RAM}} \sum_{s=1}^{S} x_{v,s}^{(t)}, \tag{7.14}$$

$$\text{STO}_v^{(t)} = \kappa_v^{\text{STO}} \sum_{s=1}^{S} x_{v,s}^{(t)}, \tag{7.15}$$

where κ_v^{CPU}, κ_v^{RAM}, and κ_v^{STO} are per-node scaling coefficients calibrated based on the processing characteristics of the node and service type.

The allocated resource vector is subject to per-node capacity constraints:

$$R_v^{\min} \leq \sum_{f \in F_v} R_{vf}^{(t)} \leq R_v^{\max} \qquad v \in V,$$

where $R_v^{\min}$ and $R_v^{\max}$ denote respectively the minimum and the maximum resource capacity vector for node v, including CPU, memory and storage. F_v denotes the set of logical functionalities $\{\text{RU}, \text{DU}, \text{CU}, \text{UPF}\}$ running on node v, and $R_{vf}^{(t)}$, the resource to be used by functionality f on node v at time t. As the system supports both vertical scaling (adjusting resources of active nodes) and horizontal scaling (activating or deactivating new instances), if a node is scaled in or out, its new configuration must also satisfy upstream and downstream capacity and latency constraints to preserve SLA guarantees.

Latency Model and SLA Constraints

In the considered 5G/B5G hierarchical architecture, each data packet transmitted from an RU traverses multiple processing layers, DU, CU, and UPF before reaching the core network. The total end-to-end latency experienced by traffic originating from an RU is written as follows:

- **Propagation Delay** (δ_{uv}^{PROP}): the transmission latency incurred when data travels over physical links between nodes u and v. These delays depend on geographical distances and transport medium characteristics (e.g., fiber optics, microwaves).

- **Processing Delay** (δ_{vf}^{PROC}): the queuing and computation delay for processing f at time t on node v, which depends on the volume of traffic handled and the amount of resources ($R_{vf}^{(t)}$) allocated to the

instance.

Let an RU r be served by a DU $d = \mathcal{P}(r)$, a CU $c = \mathcal{P}(d)$, and a UPF $u = \mathcal{P}(c)$ (the notation $\mathcal{P}(v)$ represents the parent of node v in the hierarchical topology of the network). Then, the delay $\delta_k^{(t)}$ from request k originating from RU r to UPF u is modeled as:

$$\delta_k^{(t)} = \delta_{vr}^{PROC} + \delta_{rd}^{PROP} + \delta_{vd}^{PROC} + \delta_{dc}^{PROP} + \delta_{vc}^{PROC} + \delta_{cu}^{PROP} + \delta_{vu}^{PROC}. \tag{7.16}$$

SLA Constraint: To ensure that Quality-of-Service (QoS) requirements are met, the delay for each traffic flow k originating from or destined to an active RU $r \in V_{\text{RU}}$ for slice s must not exceed a pre-defined maximum latency $L_s^{\max}$, which depends on the requested slice s:

$$\delta_k^{(t)} \leq L_s^{\max}. \tag{7.17}$$

This SLA constraint is enforced dynamically during resource scaling decisions and redirection operations. When traffic from a low-utilized RU is redirected to another active RU, the resulting end-to-end delay must also satisfy the same constraint. This guarantees that scaling decisions (both horizontal and vertical) do not lead to SLA violations.

Scaling and Control Logic

The hierarchical architecture described in the system model enables dynamic and fine-grained control of resource allocation through both *vertical* and *horizontal* scaling of processing logical functionalities deployed at RUs, DUs, CUs, and UPFs. Scaling decisions are guided by traffic forecasts and latency constraints, with the objective of minimizing energy consumption while maintaining Service Level Agreement (SLA) guarantees.

Traffic-Driven Decision Inputs: At each time step t, for every RU $r \in V_{\text{RU}}$, the forecasting model provides a probabilistic predictions, containing the point estimate, $\hat{\mathbf{I}}_r^{(t)}$, and the quantiles estimates $\hat{\mathbf{I}}_r^{(t)^{(\tau)}}$, where $\tau \in \{10, 90\}$. These represent the 10th percentile (risk-aware lower bound) and the 90th percentile (risk-aware upper bound) of the predicted traffic distribution. These probabilistic estimates are defined as:

$$\hat{\mathbf{I}}_r^{(t)} = \begin{bmatrix} \hat{x}_{r,1}^{(t+1)} & \hat{x}_{r,2}^{(t+1)} & \dots & \hat{x}_{r,S}^{(t+1)} \\ \hat{x}_{r,1}^{(t+2)} & \hat{x}_{r,2}^{(t+2)} & \dots & \hat{x}_{r,S}^{(t+2)} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{x}_{r,1}^{(t+Q)} & \hat{x}_{r,2}^{(t+Q)} & \dots & \hat{x}_{r,S}^{(t+Q)} \end{bmatrix}, \qquad \hat{\mathbf{I}}_r^{(t)^{(\tau)}} = \begin{bmatrix} \hat{x}_{r,1}^{(t+1)^{(\tau)}} & \hat{x}_{r,2}^{(t+1)^{(\tau)}} & \dots & \hat{x}_{r,S}^{(t+1)^{(\tau)}} \\ \hat{x}_{r,1}^{(t+2)^{(\tau)}} & \hat{x}_{r,2}^{(t+2)^{(\tau)}} & \dots & \hat{x}_{r,S}^{(t+2)^{(\tau)}} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{x}_{r,1}^{(t+Q)^{(\tau)}} & \hat{x}_{r,2}^{(t+Q)^{(\tau)}} & \dots & \hat{x}_{r,S}^{(t+Q)^{(\tau)}} \end{bmatrix}.$$

Given the probabilistic predictions for the RU nodes, the corresponding predictions for each node $v \in V \setminus V_{\text{RU}}$ are computed using Equation (7.12).

To reduce energy consumption, resource allocation should match predicted demand as closely as possible. However, due to forecast uncertainty, we allocate conservatively using a risk-aware effective demand. The effective demand $x_{v,s}^{(t)^{eff}}$ of node $v \in V_{\text{RU}} \cup V_{\text{DU}} \cup V_{\text{CU}} \cup V_{\text{UPF}}$ at time t for slice s is therefore computed as:

$$x_{v,s}^{(t)^{eff}} = \theta \hat{x}_{v,s}^{(t)^{(90)}} + (1-\theta)\hat{x}_{v,s}^{(t)},$$

where $\theta \in [0,1]$ controls the degree of conservativeness in provisioning. Resource allocations are then given by:

$$\hat{R}_v^{(t)} = \left[\kappa_v^{CPU} \sum_{s=1}^{S} x_{v,s}^{(t)^{eff}}, \kappa_v^{RAM} \sum_{s=1}^{S} x_{v,s}^{(t)^{eff}}, \kappa_v^{STO} \sum_{s=1}^{S} x_{v,s}^{(t)^{eff}} \right].$$

Threshold-Based Scaling Policy

Let T_v^{low} and T_v^{high} be the lower and upper traffic thresholds defining under- and over-utilization on node v. The decision engine applies the following logic:

- **Horizontal Scale-out:** If $\sum_{s=1}^{S} \hat{x}_{r,s}^{(t)^{(90)}} < T_r^{\text{low}}$, this indicates that the highest predicted traffic load at time t for RU r is below the minimum threshold required to keep it active. As a result, RU r is deactivated. Its traffic is then redirected to a neighboring active RU $\bar{r}$, chosen such that the resulting delay $\delta_k^{(t)}$ for all flows originating from or destined to $\bar{r}$ remains within the SLA constraint. When RU r is shut down, its associated DU processing instance d_r is also deactivated. This deactivation may propagate upwards: if all child DUs of a CU become inactive, the CU is deactivated as well. The same logic applies recursively from CUs to UPFs. Note that in each case, only the specific processing instances (e.g., VMs or containers) linked to the deactivated child are shut down, not the entire node, since other children might still be handling active traffic.

- **Horizontal Scale-in:** If $\sum_{s=1}^{S} \hat{x}_{r,s}^{(t)^{(10)}} \geq T_r^{\text{low}}$, and RU instance r was previously inactive, it is reactivated along with its associated DU processing instance d_r, and, if necessary, the parent CU and UPF instances.

  In RAN architectural scenarios where an RU can be connected to multiple DUs, and a DU to multiple CUs, a parent node v is only reactivated if $\sum_{s=1}^{S} \hat{x}_{v,s}^{(t,90)} \geq T_v^{\text{low}}$. Otherwise, the traffic from the child node is redirected to another already active parent node that satisfies the SLA constraint.

- **Vertical Scaling (up, or down:** For each node v, if $\sum_{s=1}^{S} x_{v,s}^{(t)^{eff}} \in [T_v^{\text{low}}, T_v^{\text{high}}]$, and the instance v is active, vertical scaling is applied by adjusting the allocated resources $R_v^{(t)}$ to match the required capacity:

$$R_v^{(t)^{\square}} = \texttt{clip}\left(\kappa_v^{\square} \sum_{s=1}^{S} x_{v,s}^{(t)^{eff}},\ R_v^{\square\,\min},\ R_v^{\square\,\max}\right), \quad \square \in \{CPU, RAM, STO\},$$

  where $\kappa_v^{\square}$ is a resource-per-traffic conversion factor, and `clip` ensures bounds are respected. $R_v^{\square\,\min}$ and $R_v^{\square\,\max}$ denote respectively the minimum and the maximum $\square$ resource capacity for node v.

  The DU, CU, and UPF processing instances linked to active children are likewise vertically scaled based on the aggregated traffic from their descendants. Note that if the aggregated effective traffic satisfies $\sum_{s=1}^{S} x_{v,s}^{(t)^{eff}} > T_v^{\text{high}}$, then the node is considered to have reached its maximum capacity, and the excess traffic is redirected to other nodes with available resources.

Each processing instance in the framework is designed to maintain stateful information, requiring consistency to be preserved during any scaling action. Horizontal deactivation is only permitted when all active child instances are either shut down or in sleep mode, ensuring that no session or state information is lost. In contrast, vertical scaling operations are preferred for preserving continuity and minimizing service disruption. This dual-scaling mechanism combining horizontal and vertical strategies enables the infrastructure to dynamically adapt to fluctuating traffic conditions while maintaining low latency. To enforce quality-of-service (QoS) guarantees, every potential scaling or redirection decision is first filtered through a latency-aware check: the resulting end-to-end delay is evaluated (see Equation (7.17)) and the operation proceeds only if the corresponding service-level agreement (SLA) constraint is satisfied. The overall control logic governing these operations is modular and scalable. It can be implemented in a distributed fashion across

different infrastructure tiers (e.g., RU, DU, CU, UPF), with centralized policy synchronization to maintain consistency. This architecture supports both real-time traffic-aware scaling and predictive provisioning based on traffic forecasts, ensuring robust and efficient system behavior.

7.4 Experimental Setup

7.4.1 Dataset Description

We evaluate our model using data traffic datasets from [Ziazet et al., 2022a; Jaumard and Ziazet, 2023]. The dataset contains time series records of traffic volume per network slice (eMBB, URLLC, mMTC) across $N = 35$ Radio Unit (RU) nodes, sampled at 15-minute intervals. Each record consists of per-slice traffic volumes measured in Mbps, resulting in multivariate time series per node. To ensure temporal continuity and avoid data leakage, we divide the dataset into training (70%), validation (10%), and test (20%) sets based on chronological order. All time series are normalized using z-score normalization on the training set statistics.

7.4.2 Baseline Methods for Comparison

To evaluate the effectiveness of our proposed ST-SplitGNN framework, we compare it to several representative baseline models. All models share the same temporal and spatio-temporal components to ensure a fair comparison. The main differences lie in data centralization, model training strategy, and spatial dependencies handling.

Baseline Methods

- **DTM**: A fully decentralized model where each node independently trains a temporal model on its local traffic time series. No spatial correlation is modeled and no information is shared between nodes.
- **CTM**: All node traffic time series are aggregated at a central location and used to train a single shared temporal model. Spatial correlations between nodes are not modeled.
- **CSTM-S**: A centralized model that incorporates both temporal modeling and spatial dependencies via a fixed, predefined adjacency matrix constructed from pairwise Pearson correlations between node time series.

- **CSTM-L**: A centralized model where the adjacency matrix is learned jointly with the model parameters during training, enabling adaptive spatial correlation learning across nodes.

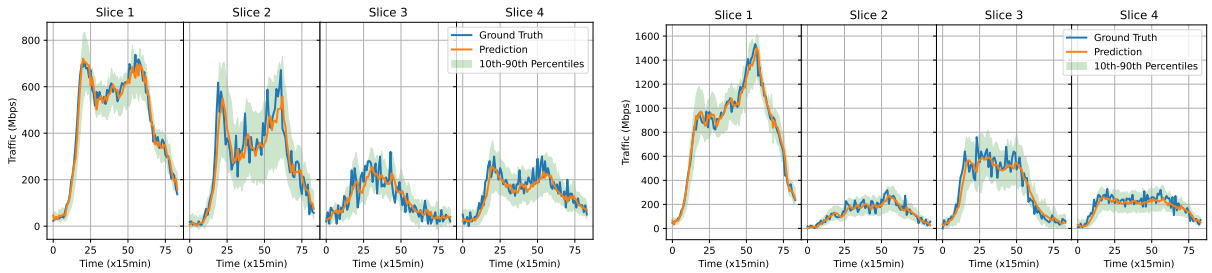
Newly Proposed Methods

- **Split Spatio-Temporal Model with Static Graph (ST-SplitGNN-S)**: Our split learning framework where each node trains a local temporal encoder, and the server aggregates latent representations using a GNN with a static, predefined adjacency matrix (Pearson correlation-based). This setup captures spatial dependencies while enabling temporal model specialization.
- **Split Spatio-Temporal Model with Learnable Graph (ST-SplitGNN)**: The full version of our proposed framework. Clients train temporal encoders locally, while the server-side GNN learns both the spatial dependencies (adjacency matrix) and aggregation logic, allowing for improved adaptability and performance across varying traffic patterns.

We assess predictive model performance using the following standard regression metrics: Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the Mean Absolute Percentage Error (MAPE). Furthermore, the proposed uncertainty-aware resource allocation scheme, named here **ST-SplitGNN+**, is evaluated against two baseline methods. The first, **PeakRes**, allocates the maximum resources observed during peak hours at a given node in training time at each time step. The second, **ST-SplitGNN-**, is a variant of our proposed approach that considers only the predicted traffic, without accounting for uncertainty. We assess the performance of the three approaches, **PeakRes**, **ST-SplitGNN-**, and **ST-SplitGNN+** in terms of SLA violation, resource utilization, and energy efficiency. Results are computed per node and over time across all test samples. Each metric highlights a key trade-off in managing uncertainty, resource provisioning, and service reliability.

7.4.3 Implementation Details

The client-side temporal encoders are implemented using 3-layer LSTM networks with hidden size 64. The server-side spatio temporal block consists of 2 MTGNN layers (as in [Wu et al., 2020b]) with 2 GCN blocks. The learnable adjacency matrix is initialized using randomly initialized node embeddings and updated during training via backpropagation. The model is trained for 50 epochs using Adam optimizer with learning rate 0.001 and batch size 32. The window size is set to $T = 8$, with a prediction horizon of



(a) RU 2

(b) RU 4

Figure 7.3: Actual vs. predicted traffic volumes over time for two representative nodes across different slices. The plots compare the ground truth traffic with predictions from the proposed ST-SplitGNN model. Across both nodes and slice types, ST-SplitGNN accurately tracks temporal trends with minimal lag and strong alignment, demonstrating its effectiveness in capturing both local and global traffic dynamics.

$Q = 1$. GNN residual weight α is set to 0.1, and the adjacency saturation parameter β to 3. Node CPU, memory, storage capacity, and delay limit as well as the resource-per-traffic conversion factor κ_r are chosen as in [Ziazet et al., 2023].

7.5 Performance Evaluation

This section presents a comprehensive evaluation of our proposed framework. The evaluation is organized into two main parts: the first focuses on assessing the quality of the traffic prediction model, while the second examines the effectiveness of the resource allocation scheme. Together, these analyses demonstrate the benefits of incorporating spatio-temporal learning and uncertainty awareness into resource management decisions.

7.5.1 Evaluation of the Predictive Model

In this subsection, we assess the performance of the proposed ST-SplitGNN+ against baseline models.

Quantitative Results of Prediction

Table 7.1 summarizes the test set performance of all evaluated models across Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE) metrics (lower values indicate better accuracy). The fully decentralized temporal model (DTM) and the centralized temporal model (CTM) exhibit the weakest performance, with the highest error rates across all three metrics. This

highlights the limitations of relying solely on local temporal patterns or centralized models that lack spatial context.

Introducing spatial modeling with a predefined adjacency matrix yields noticeable improvements. Specifically, both the centralized spatio-temporal model with static adjacency (CSTM-S) and its learnable version (CSTM-L) reduce all error metrics significantly, confirming the benefit of incorporating spatial dependencies. Among them, CSTM-L performs better than CSTM-S, validating the effectiveness of learning the adjacency matrix rather than relying on fixed correlations.

Our proposed ST-SplitGNN models, which leverage split learning with per-node specialization and server-side spatial aggregation, outperform all centralized and decentralized baselines. Notably, ST-SplitGNN-S surpasses all centralized models, demonstrating the benefit of node-specific temporal modeling even when using a static graph. This specialization captures recurring and node-specific traffic trends, which are often smoothed out in fully centralized models. As a result, our client-side encoders are more sensitive to temporal nuances and bursty behaviors that may be lost in shared representations. The best overall performance is achieved by our ST-SplitGNN model, which combines specialized local temporal encoding with learnable spatial dependencies, achieving a substantial reduction in MAE (↓25.60), RMSE (↓32.55), and MAPE (↓25.5%). These results clearly demonstrate the advantage of the spatio-temporal split learning paradigm, particularly when equipped with a learnable graph structure that adapts to real traffic dependencies for enhanced prediction accuracy.

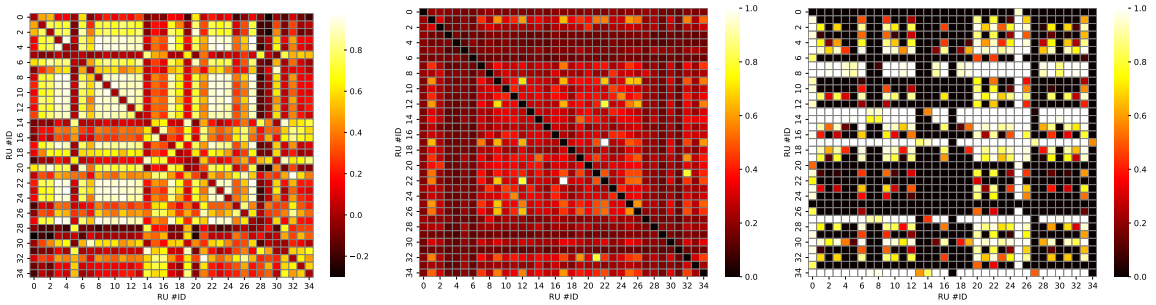
Table 7.1: Traffic prediction performance comparison.

Model	**MAE ↓**	**RMSE ↓**	**MAPE ↓**
Decentralized Temporal Model (DTM)	45.31	58.42	79.2%
Centralized Temporal Model (CTM)	46.95	60.87	82.2%
Centralized Spatio-Temporal Model with Static Graph (CSTM-S)	35.82	47.23	50.5%
Centralized Spatio-Temporal Model with Learnable Graph (CSTM-L)	33.12	44.56	47.3%
ST-SplitGNN-S	30.61	40.68	33.4%
ST-SplitGNN	**25.60**	**32.55**	**25.5%**

Lower MAE, RMSE and MAPE are better.

Effectiveness of Learnable Graph Aggregation

As shown in Table 7.1 , our results show that ST-SplitGNN, and CSTM-L respectively consistently outperforms ST-SplitGNN-S and CSTM-S, demonstrating the benefit of jointly learning the adjacency matrix



(a) Pearson correlation adjacency matrix. (b) DTW correlation adjacency matrix. (c) Learned adjacency matrix.

Figure 7.4: Comparison of node adjacency matrices for spatial dependency modeling. The heatmaps show (a) Pearson correlation-based, (b) Dynamic Time Warping (DTW)-based, and (c) the learned adjacency matrix from our proposed ST-SplitGNN framework. The learned matrix is notably sparser and more adaptive, selectively capturing the most relevant and possibly asymmetric or nonlinear dependencies between nodes. Unlike fixed, symmetric correlation-based methods, it is data-driven and task-optimized, enabling better modeling of traffic interactions for accurate prediction.

during training. By allowing the model to dynamically adapt spatial relationships based on the prediction objective, ST-SplitGNN avoids incorporating spurious or irrelevant dependencies that are often present in static correlation-based graphs. Furthermore, the learned adjacency matrix is notably more sparse and asymmetric, as visualized in Figure 7.4, indicating that the model selects only the most informative neighbors for each node. This sparsity likely reduces noise propagation across unrelated nodes and improves the robustness of traffic prediction, especially in heterogeneous network environments where node behavior may differ significantly. Asymmetry may come, e.g., from forward/backward waves (residential vs. downtown areas) as observed in real roadways [Hwang et al., 2022], or from role played by game servers vs. game players.

Temporal Prediction Accuracy Visualization

Figure 7.3 presents the actual versus predicted traffic volumes for two representative nodes across different service slices, as output by the ST-SplitGNN model. The plots demonstrate that ST-SplitGNN closely tracks the evolving traffic dynamics, effectively capturing periodic patterns, sudden spikes, and gradual fluctuations. This strong alignment across both nodes and slice types highlights the model's ability to learn both slice-specific and node-specific temporal characteristics. The close match between predicted and actual values confirms the model's generalization capability across diverse traffic profiles, reinforcing its suitability for real-time traffic forecasting in multi-slice 5G networks.

Scalability and Practicality

From a systems perspective, split learning reduces the communication burden and enhances scalability. Nodes only transmit activations rather than raw data, reducing privacy concerns and enabling training in bandwidth-constrained environments. Furthermore, the modular design allows seamless updates or replacements of client models without requiring retraining of the entire system.

Limitations and Future Directions

One limitation of the current setup is the assumption of synchronized time windows and uniform data availability across nodes. Future work could investigate asynchronous or missing-data scenarios. Additionally, extending this framework to incorporate inter-slice dependencies or multi-hop graph propagation could further boost prediction accuracy and adaptability.

7.5.2 Evaluation of the Resource Allocation Scheme

This subsection evaluates the effectiveness of the proposed uncertainty-aware resource allocation strategy in terms of three key metrics: SLA violation, resource utilization, and energy efficiency.

SLA Violation Analysis

To evaluate service reliability under varying traffic conditions, we measure the SLA violation ratio as the proportion of traffic demand that is not served. At each node and time step, the violation is computed as:

$$\text{SLA Violation} = 100 \times \left(1 - \frac{\text{Accepted Traffic}}{\text{Total Traffic Demand}}\right). \tag{7.18}$$

This metric captures how much of the requested bandwidth was denied due to insufficient allocated resources. A value of 0 indicates full compliance with the SLA (no traffic loss), while values closer to 100% indicate severe underprovisioning or service outage. Figure 7.5 presents the average SLA violation ratio per node across time for the three compared approaches. PeakRes shows near-perfect SLA compliance with an average violation close to 0%. However, we observe small but non-zero violations ($\geq 0.5\%$) in certain nodes. This is likely due to the fact that the peak resource profiles are estimated based on training data only. During testing, nodes that experience sudden surges in traffic beyond the historical peak can exceed the allocated capacity, leading to occasional SLA breaches. ST-SplitGNN-, which relies solely on point

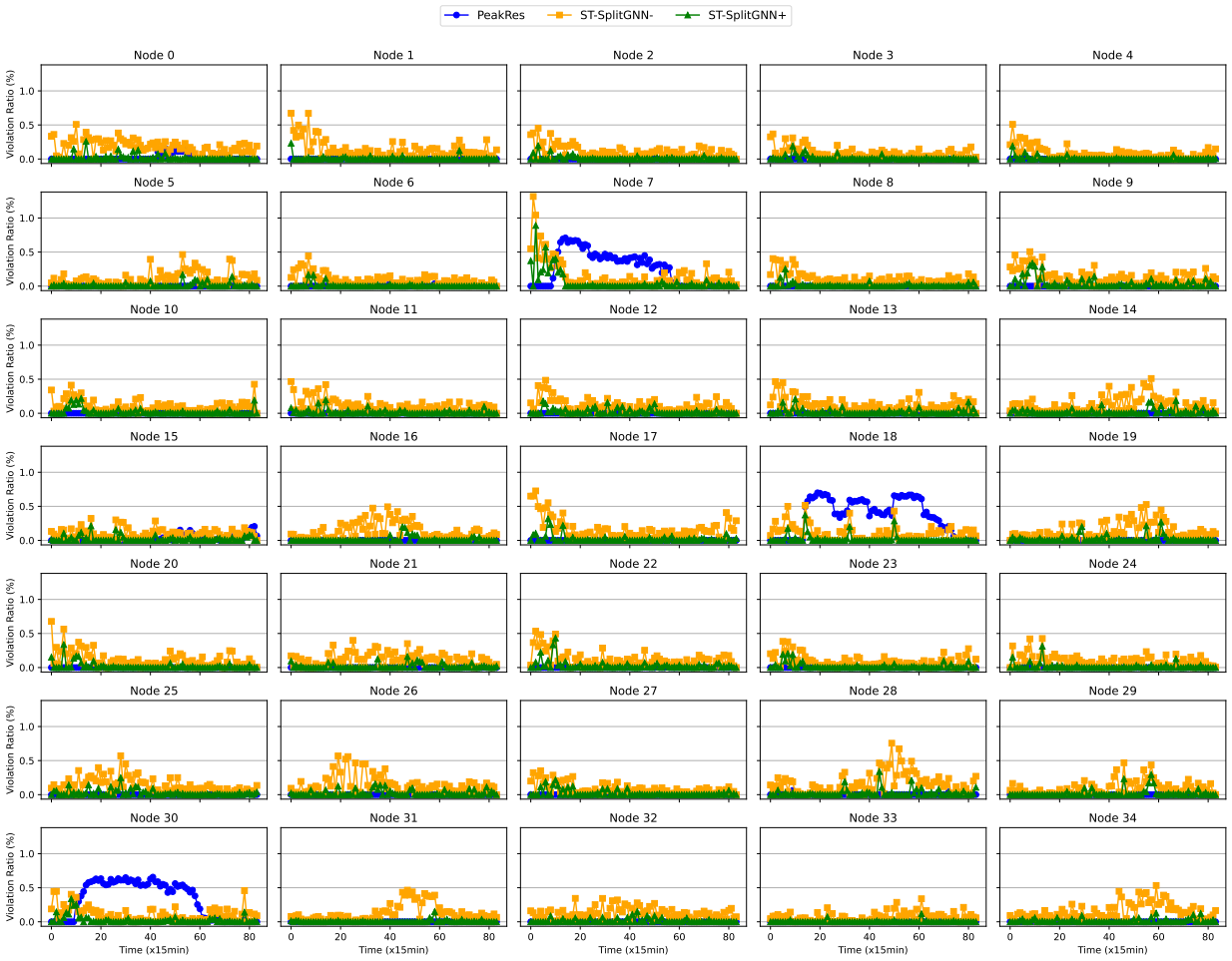


Figure 7.5: SLA violation ratio. PeakRes achieves near-zero violations due to aggressive over-provisioning, but shows occasional violations in some nodes where traffic exceeds training-time peaks. ST-SplitGNN- suffers the highest violation rates due to its tight allocation around predictions. The proposed ST-SplitGNN+ reduces violations significantly by incorporating uncertainty, achieving a balanced trade-off between reliability and efficiency.

traffic predictions without accounting for uncertainty, exhibits more frequent SLA violations. This under-provisioning stems from prediction errors that underestimate the actual traffic, thus allocating insufficient resources. This highlights the risk of relying solely on mean predictions for provisioning. ST-SplitGNN+, our proposed uncertainty-aware approach, mitigates this risk by incorporating traffic uncertainty into its resource allocation decisions. As a result, it achieves a balanced SLA violation profile, significantly better than ST-SplitGNN-, although it occasionally experiences minor violations comparable to PeakRes.

This clearly highlights the importance of incorporating uncertainty into resource provisioning. While PeakRes ensures high reliability, it does so at the cost of overprovisioning. On the other hand, ST-SplitGNN+ offers a strong trade-off by ensuring high reliability while avoiding excessive resource usage.

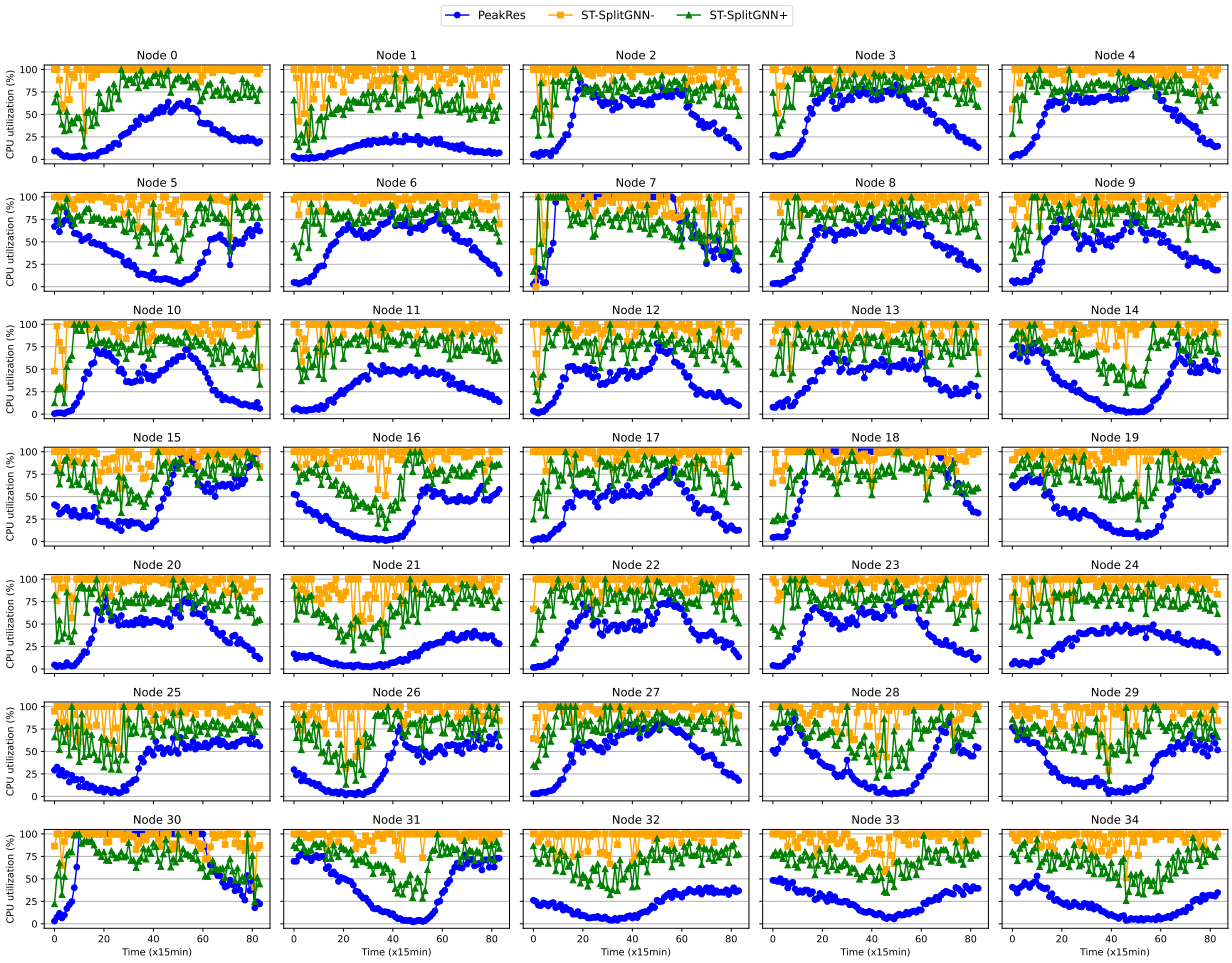


Figure 7.6: CPU resource utilization across RU nodes. PeakRes shows consistently low utilization due to over-provisioning, leading to resource underuse. ST-SplitGNN- achieves high utilization by tightly following traffic predictions, but risks under-provisioning. The proposed ST-SplitGNN+ approach maintains a balanced utilization profile, reducing idle capacity while preserving robustness to traffic uncertainty.

Resource Utilization Analysis

Here, we evaluate how effectively each approach uses the allocated resources over time. Utilization is measured as the ratio between actual demand and allocated capacity per resource type (CPU, memory, storage). For brevity, we report only CPU utilization, as memory and storage exhibit similar patterns. As illustrated in Figure 7.6, the PeakRes approach consistently exhibits low resource utilization. This is a direct consequence of its over-provisioning strategy, which allocates capacity based on historical traffic peaks. While this ensures minimal SLA violations, it results in significant idle capacity, especially during low-traffic periods. In contrast, ST-SplitGNN- achieves high utilization by provisioning tightly around point traffic predictions. However, this aggressive allocation strategy increases the risk of SLA violations

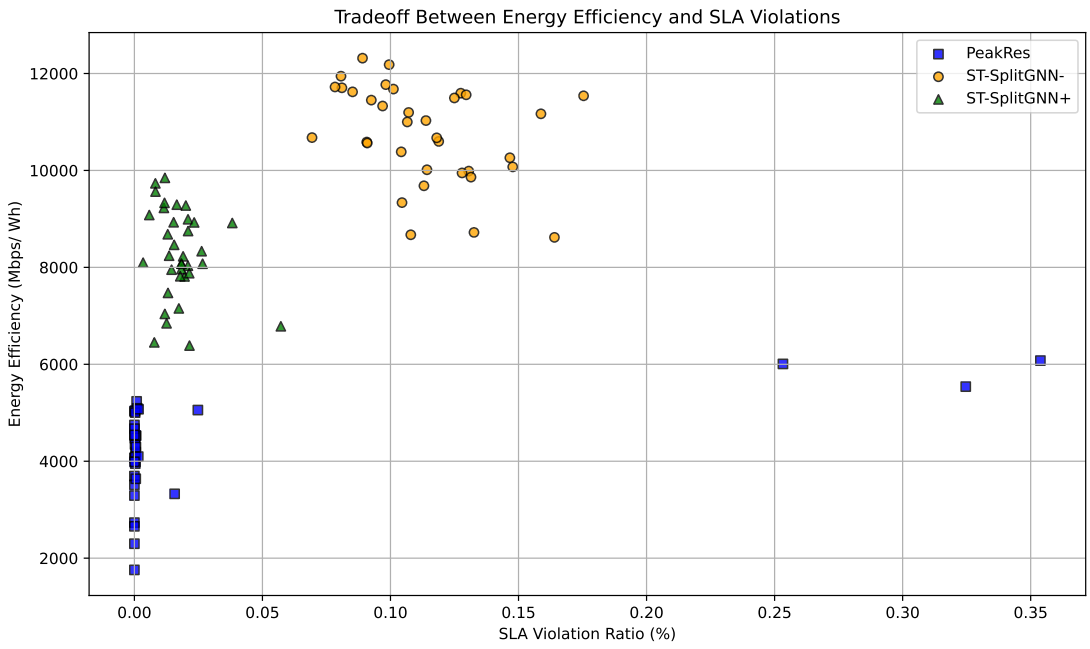


Figure 7.7: Trade-off between SLA violation and energy efficiency. ST-SplitGNN- achieves the highest energy efficiency but suffers from high SLA violations. PeakRes ensures minimal SLA violations but at the cost of poor energy efficiency. ST-SplitGNN+ balances the trade-off, achieving significant energy savings while maintaining acceptable SLA performance.

due to prediction errors and lack of safety margins, as discussed previously. The proposed ST-SplitGNN+ approach strikes a balance between efficiency and reliability. By incorporating traffic uncertainty into its scaling decisions, it reduces idle capacity compared to PeakRes while avoiding the underprovisioning issues seen in ST-SplitGNN-. Across all nodes, ST-SplitGNN+ maintains utilization levels between the two baselines, confirming its ability to adapt resource allocation based on both predicted demand and its associated uncertainty.

Energy Efficiency vs. SLA Violation Trade-off

In this analysis, we examine the trade-off between energy efficiency and SLA satisfaction for each of the three approaches. Energy efficiency is measured in Mbps/Wh, while SLA violation is quantified as the ratio of unserved traffic over total demand. Figure 7.7 presents this trade-off by plotting energy efficiency against SLA violation with each point representing a node. As shown in the figure, ST-SplitGNN- achieves the highest energy efficiency. This outcome is expected, as it allocates resources tightly around predicted traffic, minimizing energy waste due to idle capacity. However, this aggressive provisioning strategy results

in the highest SLA violation among the three methods. This underscores the vulnerability of ST-SplitGNN- to demand underestimation, particularly in the presence of traffic fluctuations. On the other end of the spectrum, PeakRes exhibits nearly zero SLA violations as discussed previously, thanks to its conservative over-provisioning based on peak traffic levels. This robustness, however, comes at the expense of energy efficiency, which is significantly lower due to persistent over-allocation, even during periods of low demand. The proposed ST-SplitGNN+ offers a compelling middle ground. By explicitly accounting for prediction uncertainty, it introduces a flexible buffer in resource provisioning, leading to a balanced trade-off between energy efficiency and SLA adherence. Its performance lies between the two baselines, demonstrating both substantial energy savings compared to PeakRes and significantly fewer SLA violations than ST-SplitGNN-.

This result highlights the strength of our approach: ST-SplitGNN+ not only adapts to temporal traffic dynamics but also manages resource allocation in a way that improves sustainability without compromising reliability. As such, it presents a promising strategy for intelligent and energy-aware management in future 5G and beyond networks.

7.6 Conclusion

In this chapter, we proposed a two-stage framework for intelligent traffic prediction and resource management in 5G and beyond networks. ST-SplitGNN introduces a spatio-temporal split learning approach that combines local temporal encoders with a centralized GNN to capture both node-level patterns and inter-node dependencies. Building on this, ST-SplitGNN+ uses point forecasts and uncertainty estimates to enable proactive, uncertainty-aware resource allocation. Our approach outperforms baselines in terms of prediction accuracy, reveals meaningful spatial relationships, and ensures scalable, privacy-preserving learning. The resource allocation strategy adapts to both predicted demand and its uncertainty, improving sustainability and reliability by avoiding over- or under-provisioning. This framework offers a robust foundation for energy-efficient, SLA-compliant network management.

Chapter 8

Conclusion and Future Work

This thesis aims to address a critical and timely challenge: enabling *cognitive and sustainable communication networks* by unifying data-centric modeling, energy-aware optimization, distributed learning, and predictive control. Through a series of interconnected contributions, the thesis constructs a layered architecture where each chapter builds upon the previous, forming a coherent and actionable framework for the next generation of intelligent networks.

At the foundation, we tackled a fundamental bottleneck in AI-driven networking: the lack of high-quality, representative available datasets. By developing techniques for both packet-level simulation-based data generation and flow-level real-world data refactoring, it offers a dual strategy to overcome challenges related to data scarcity, heterogeneity, and noise. The former, rooted in data-centric AI, generates simulation-based datasets that capture nuanced network behavior. The latter reconstructs flow-level data from real-world urban mobility traces, preserving spatio-temporal patterns essential for model robustness. The key contribution lies not just in producing datasets, but in showcasing how data- centric design can tailor ML models to the nuanced dynamics of communication networks. It promotes a shift towards *data-centric AI* in networking, which is often overshadowed by model-centric efforts. The real impact is enabling reproducible and domain-specific AI development, setting a methodological precedent for the community.

On this foundation, we addressed the joint problem of logical function placement (DU/CU/UPF) and resource provisioning and routing, a key enabler of energy-aware and latency-sensitive 5G and beyond architectures. Our decomposition-based column generation algorithm offers near-optimal solutions, while our heuristic method maintains high performance with far greater scalability. The proposed solutions not only reduce energy consumption by up to 14% but also deliver low-latency performance. This work highlights

that careful design and decomposition in network optimization can deliver real sustainability gains. It also emphasizes the pitfalls of static peak-hour planning, advocating for more better holistic temporal strategies in infrastructure deployment.

In the realm of learning algorithms, as network devices are distributed by nature, we introduced AFSL and AFSL+, asynchronous federated split learning protocols tailored for heterogeneous and non-IID edge environments. AFSL harmonizes federated and split learning to mitigate the impact of stragglers, device diversity, accelerating convergence by up to 86% without sacrificing accuracy. AFSL+ builds on this by incorporating a novel clustering-based client selection mechanism that balances energy profiles, data representativeness, and performance. Together, they mark a significant departure from synchronous schemes that dominate the literature, offering a more adaptable and sustainable solution for real-world deployment in IoT and mobile networks. The solution presented in this chapter expand the feasibility of distributed ML in non-ideal environments, which are the norm rather than the exception in B5G networks. It shifts the design paradigm from idealized synchrony to practical adaptability. It also suggests that energy-efficient learning is not just about preserving device energy consumption, it is about enabling continuous, fair and sustainable intelligence at the network edge.

Further, as communication overhead often overshadow compute limitations in distributed learning, we advanced the field of communication-efficient learning by proposing a dynamic compression technique tailored for split learning. Unlike static schemes, our method adapts compression in real-time to fluctuating bandwidth conditions. The result is a protocol that intelligently reduces data transmission by up to 82% while maintaining, or even enhancing, model generalization. This adaptive compression adds a new layer of flexibility and efficiency, extending the operational feasibility of split learning in constrained environments such as IoT networks and mobile systems. It does not only solve a practical challenge, but also opens new perspectives on compression as a form of regularization.

Finally, in the most integrative layer of the thesis, we presented ST-SplitGNN and ST-SplitGNN+, spatio-temporal split learning frameworks for traffic prediction and SLA-compliant resource allocation. These models combine temporal encoders, learned spatial graph modeling, and uncertainty-aware decision making to deliver high accuracy predictions and smarter provisioning decisions. The resulting system is not only reactive but predictive, enabling cognitive control where learning directly informs orchestration. It proves that ML in networking can move beyond passive modeling to active, closed-loop decision making. In an age of resource scarcity and variable demand, this is the kind of intelligence networks must have.

In essence, these contributions form an end-to-end ecosystem: data generation as the fertile ground, optimization as the infrastructure, distributed learning as the computational engine, and predictive control as the actuator. Together, these layers yield a self-sustaining, intelligent network that can adapt, learn, and optimize under uncertainty and constraints. This metaphor captures not just the architecture, but the philosophy behind the work: sustainable intelligence through systemic integration. Importantly, these contributions are not just academic. They resonate with pressing industrial and societal needs. As telcos and governments commit to green networking, carbon budgeting, and data privacy compliance, the methods developed here offer timely solutions. From enabling energy-aware federated training in IoT, to adaptive provisioning under SLA and bandwidth constraints, this thesis lays technical groundwork for the AI-native, sustainable infrastructures envisioned for 6G.

Looking ahead, several promising research directions emerge. First, continual and online learning should be integrated into AFSL+ to enable model updates in response to evolving traffic, user churn, and shifting workloads. Second, incorporating transformer architectures into our predictive control and compression modules may improve the modeling of long-range dependencies. Third, the ST-SplitGNN framework could evolve into a self-adaptive orchestration engine by integrating runtime feedback loops, enabling models to self-tune under operational constraints. Fourth, the joint optimization of energy-latency-accuracy trade-offs across the entire ML pipeline spanning data generation, model placement, training, communication, and inference, presents a grand challenge that remains open. Finally, uncertainty-aware control could be deepened through reinforcement learning or Bayesian decision-making, allowing resource allocation to become more resilient in the face of fluctuating demand and volatile network conditions.

In sum, this thesis offers both a blueprint and a vision. It presents a unified, operational framework where intelligent networks are not merely reactive or high-performing, but sustainable, adaptive, and self-aware. This shows that sustainability and intelligence are not opposing goals, but synergistic ones when designed with an end-to-end perspective. As we move into an era where data volume, energy constraints, model complexity, and real-time demands converge, the future of networking lies not just in faster connections but in *cognitively connected systems*, networks that learn, adapt, and sustain themselves. The journey towards that vision continues.

Bibliography

3GPP (2018a). 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; NG-RAN; Architecture Description (release 15). Version 15.2.0.

3GPP (2018b). 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Study on Management and Orchestration of Network Slicing for Next Generation Network (Release 15). Version 15.1.0.

3GPP (2018c). 5G - Procedures for the 5G System. 3GPP TS 23.502 version 15.2.0 Release 15.

3GPP (2021). System Architecture for the 5G system (5GS). TS 23.501, Version 17.3.0.

3GPP (2022). 5G; architecture enhancements for 5G system (5GS) to support network data analytics services. 3GPP TS 23.288 version 17.5.0 Release 17.

3GPP (Dec. 2024a). Architecture enhancements for 5G system (5GS) to support network data analytics services. *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.288, version 18.8.0.*

3GPP (Dec. 2024b). Study on core network enhanced support for artificial intelligence (AI) / machine learning (ML). *3rd Generation Partnership Project (3GPP), Technical Specification (TS) 23.700-84, version 19.0.0.*

Aceto, G., Ciuonzo, D., Montieri, A., and Pescapé, A. (2019). Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. *IEEE transactions on network and service management*, 16(2):445–458.

Ahvar, E., Orgerie, A.-C., and Lebre, A. (2022). Estimating energy consumption of cloud, fog, and edge computing infrastructures. *IEEE Transactions on Sustainable Computing*, 7(2):277–288.

Al-Karawi, Y., Al-Raweshidy, H., and Nilavalan, R. (2024). Power consumption evaluation of next generation open radio access network. In *IEEE International Conference on Consumer Electronics (ICCE)*, pages 1–6.

Albuquerque, R., Dias, L., Ziazet, M., Vandikas, K., Ickin, S., Jaumard, B., Natalino, C., Wosinska, L., Monti, P., and Wong, E. (2024). Asynchronous federated split learning. In *2024 IEEE International Conference on Fog and Edge Computing (ICFEC)*, pages 11–18.

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. (2017). QSGD: Communication-efficient SGD via gradient quantization and encoding. In *International Conference on Neural Information Processing Systems (NIPS)*, volume 30, pages 1707 – 1718.

Almanifi, O. R. A., Chow, C.-O., Tham, M.-L., Chuah, J. H., and Kanesan, J. (2023). Communication and computation efficiency in federated learning: A survey. *Internet of Things*, 22:100742.

Amiri, M. and Mohammad-Khanli, L. (2017). Survey on prediction models of applications for resources provisioning in cloud. *Journal of Network and Computer Applications*, 82:93–113.

Amor, H. B., Desrosiers, J., and Frangioni, A. (2004). Stabilization in column generation. *Les Cahiers du GERAD ISSN*, 711:2440.

An, C., Lu, Z., Ma, Y., Zhao, W., and Chen, X. (2025). Prediction of multivariate spatial-temporal series data based on adaptive spatial-temporal information. *IEEE Transactions on Big Data*.

Andrew, N., Dillon, L., and Lynn, H. (2021). Data-centric AI Competition. `https://https-deeplearning-ai.github.io/data-centric-comp/`.

Aouedi, O., Le, V. A., Piamrat, K., and Ji, Y. (2025). Deep learning on network traffic prediction: Recent advances, analysis, and future directions. *ACM Computing Surveys*, 57(6):1–37.

Arouj, A. and Abdelmoniem, A. M. (2024). Towards energy-aware federated learning via collaborative computing approach. *Computer Communications*, 221:131–141.

Askari, L., Hmaity, A., Musumeci, F., and Tornatore, M. (2018). Virtual-network-function placement for dynamic service chaining in metro-area networks. In *International Conference on Optical Network Design and Modeling (ONDM)*, pages 136 – 141, Dublin, Ireland.

Askari, L., Musumeci, F., Salerno, L., Ayoub, O., and Tornatore, M. (2020). Dynamic DU/CU placement for 3-layer C-RANs in optical metro-access networks. In *International Conference on Transparent Optical Networks (ICTON)*, pages 1–4.

Ayad, A., Renner, M., and Schmeink, A. (2021). Improving the communication and computation efficiency of split learning for IoT applications. In *IEEE Global Comm. Conf. (GLOBECOM)*, pages 1 – 6.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. In *International Conference on Learning Representations (ICLR)*, pages 1–15.

Bahreini, T., Badri, H., and Grosu, D. (2019). Energy-aware capacity provisioning and resource allocation in edge computing systems. In *Edge Computing (EDGE)*, pages 31–45.

Bari, F., Chowdhury, S. R., Ahmed, R., Boutaba, R., and Duarte, O. C. M. B. (2016). Orchestrating virtualized network functions. *IEEE Transactions on Network and Service Management*, 13(4):725–739.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329.

Benzekki, K., El Fergougui, A., and Elbelrhiti Elalaoui, A. (2016). Software-defined networking (sdn): a survey. *Security and communication networks*, 9(18):5803–5833.

Bertsimas, D. and Tsitsiklis, J. N. (1997). *Introduction to linear optimization*, volume 6.

Betker, A., Gerlach, C., Hülsermann, R., Jäger, M., Barry, M., Bodamer, S., Späth, J., Gauger, C., and Köhn, M. (July 2003). Reference transport network scenarios. Technical report, BMBF Multi Tera Net.

Bi, Z., Yu, L., Gao, H., Zhou, P., and Yao, H. (2021). Improved VGG model-based efficient traffic sign recognition for safe driving in 5G scenarios. *International Journal of Machine Learning and Cybernetics*, 12:3069–3080.

Bianzino, A. P., Chaudet, C., Rossi, D., and Rougier, J.-L. (2010). A survey of green networking research. *IEEE Communications Surveys & Tutorials*, 14(1):3–20.

Blanco, B., Fajardo, J. O., Giannoulakis, I., Kafetzakis, E., Peng, S., Pérez-Romero, J., Trajkovska, I., Khodashenas, P. S., Goratti, L., Paolino, M., et al. (2017). Technology pillars in the architecture of future 5G mobile networks: NFV, MEC and SDN. *Computer Standards & Interfaces*, 54:216–228.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308.

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, B., et al. (2019). Towards federated learning at scale: System design. *Proceedings of machine learning and systems*, 1:374–388.

Brik, B., Boutiba, K., and Ksentini, A. (2022). Deep learning for B5G open radio access network: Evolution, survey, case studies, and challenges. *IEEE Open Journal of the Communications Society*, 3:228–250.

Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A., and Vandergheynst, P. (2017). Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42.

Cappanera, P., Paganelli, F., and Paradiso, F. (2019). Vnf placement for service chaining in a distributed cloud environment with multiple stakeholders. *Computer Communications*, 133:24–40.

Chai, Z., Chen, Y., Anwar, A., Zhao, L., Cheng, Y., and Rangwala, H. (2021). FedAT: A high-performance and communication-efficient federated learning system with asynchronous tiers. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16.

Chen, C., Liu, Y., Ma, X., and Lyu, L. (2022). CALFAT: Calibrated federated adversarial training with label skewness. *Advances in Neural Information Processing Systems*, 35:3569–3581.

Chen, M., Challita, U., Saad, W., Yin, C., and Debbah, M. (2019a). Artificial neural networks-based machine learning for wireless networks: A tutorial. *IEEE Communications Surveys & Tutorials*, 21(4):3039–3071.

Chen, M., Miao, Y., Gharavi, H., Hu, L., and Humar, I. (2019b). Intelligent traffic adaptive resource allocation for edge computing-based 5G networks. *IEEE transactions on cognitive communications and networking*, 6(2):499–508.

Chen, X., Li, J., and Chakrabarti, C. (2021a). Communication and computation reduction for split learning using asynchronous training. In *IEEE Workshop on Signal Processing Systems (SiPS)*, pages 76–81.

Chen, X., Li, J., and Chakrabarti, C. (2021b). Communication and computation reduction for split learning using asynchronous training. In *IEEE Workshop on Signal Processing Systems (SiPS)*, pages 76–81.

Chen, Y., Ning, Y., Slawski, M., and Rangwala, H. (2020). Asynchronous online federated learning for edge devices with non-IID data. In *IEEE International Conference on Big Data (Big Data)*, pages 15–24.

Chopra, A., Sahu, S. K., Singh, A., Java, A., Vepakomma, P., Amiri, M. M., and Raskar, R. (2023). Adaptive split learning. In *Federated Learning Systems (FLSys) Workshop@ MLSys*, pages 1 – 13.

Chvatal, V. (1983). *Linear Programming*. Freeman.

Corcoran, D., Kreuger, P., and Schulte, C. (2020). Efficient real-time traffic generation for 5G RAN. In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–9.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2022). *Introduction to algorithms*. MIT press.

Cui, Z., Zhang, J., Noh, G., and Park, H. J. (2023). Adstgcn: A dynamic adaptive deeper spatio-temporal graph convolutional network for multi-step traffic forecasting. *Sensors*, 23(15):6950.

Dantzig, G. B. (1948). Programming in a linear structure. *Washington, DC*.

Dantzig, G. B. (2016). *Linear programming and extensions*. Princeton university press.

Dataset, M. (2022). Comptages des véhicules, cyclistes et piétons aux intersections munies de feux de circulation. `https://donnees.montreal.ca/ville-de-montreal/comptage-vehicules-pietons`. Accessed: 2022-04-06.

Deinlein, T., German, R., and Djanatliev, A. (2020). 5G-Sim-V2I/N: towards a simulation framework for the evaluation of 5G V2I/V2N use cases. In *European Conference on Networks and Communications (EuCNC)*, pages 353–357. IEEE.

Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2006). *Column generation*, volume 5. Springer Science & Business Media.

Desrosiers, J. and Lübbecke, M. E. (2005). A primer in column generation. In *Column generation*, pages 1–32. Springer.

Ding, Z., Wang, S., and Jiang, C. (2023). Kubernetes-oriented microservice placement with dynamic resource allocation. *IEEE Transactions on Cloud Computing*, 11(2):1777–1793.

Duan, Q. et al. (2022). Combined federated and split learning in edge computing for ubiquitous intelligence in internet of things: State-of-the-art and future directions. *Sensors*, 22(16):5983.

Dwaraki, A. and Wolf, T. (2016). Adaptive service-chain routing for virtual network functions in software-defined networks. In *Workshop on Hot topics in Middleboxes and Network Function Virtualization (HotMIddlebox)*, pages 32–37.

Eisen, M., Zhang, C., Chamon, L. F., Lee, D. D., and Ribeiro, A. (2019). Learning optimal resource allocations in wireless systems. *IEEE Transactions on Signal Processing*, 67(10):2775–2790.

Ericsson (2024). Ericsson mobility report november 2024 [online]. `https://www.ericsson.com/en/reports-and-papers/mobility-report/reports`.

ETSI (2014). Network functions virtualisation (NFV); architectural framework. ETSI GS NFV 002 V1.2.1.

ETSI, G. N. . (2018). Next generation protocols (NGP); E2E network slicing reference framework and information model. [online].

Feng, W., Zhang, J., Dong, Y., Han, Y., Luan, H., Xu, Q., Yang, Q., Kharlamov, E., and Tang, J. (2020). Graph random neural networks for semi-supervised learning on graphs. *Advances in neural information processing systems*, 33:22092–22103.

Ferriol-Galmés, M., Paillisse, J., Suárez-Varela, J., Rusek, K., Xiao, S., Shi, X., Cheng, X., Barlet-Ros, P., and Cabellos-Aparicio, A. (2023a). Routenet-fermi: Network modeling with graph neural networks. *IEEE/ACM Transactions on Networking*, pages 1–0.

Ferriol-Galmés, M., Paillisse, J., Suárez-Varela, J., Rusek, K., Xiao, S., Shi, X., Cheng, X., Barlet-Ros, P., and Cabellos-Aparicio, A. (2023b). Routenet-fermi: Network modeling with graph neural networks. *IEEE/ACM transactions on networking*, 31(6):3080–3095.

Flinta, C., Yan, W., and Johnsson, A. (2020). Predicting round-trip time distributions in IoT systems using histogram estimators. In *IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9.

Flunkert, V., Salinas, D., and Gasthaus, J. (2017). Deepar: Probabilistic forecasting with autoregressive recurrent networks. *arXiv preprint arXiv:1704.04110*, 23.

Foukas, X., Nikaein, N., Kassem, M. M., Marina, M. K., and Kontovasilis, K. (2016). FlexRAN: A flexible and programmable platform for software-defined radio access networks. In *International on Conference on emerging Networking EXperiments and Technologies*, pages 427–441.

Foukas, X., Patounas, G., Elmokashfi, A., and Marina, M. K. (2017). Network slicing in 5G: Survey and challenges. *IEEE communications magazine*, 55(5):94–100.

Fu, L., Zhang, H., Gao, G., Zhang, M., and Liu, X. (2023). Client selection in federated learning: Principles, challenges, and opportunities. *IEEE Internet of Things Journal*, 10(24):21811–21819.

Gao, Y., Kim, M., Thapa, C., Abuadbba, A., Zhang, Z., Camtepe, S., Kim, H., and Nepal, S. (2021). Evaluation and optimization of distributed machine learning techniques for internet of things. *IEEE Transactions on Computers*, 71(10):2538–2552.

Ghazizadeh, A., Akbari, B., and Tajiki, M. M. (2022). Joint reliability-aware and cost efficient path allocationfig and VNF placement using sharing scheme. *Journal of Network and Systems Management*, 30:1–28.

gnnet2022 (2022). BNN-UPC graph neural networking challenge 2022. https://bnn.upc.edu/challenge/gnnet2022/.

GSMA-Intelligence (2023). IoT market forecast to 2030: connections by region and vertical [online]. `http://gsmaintelligence.com/research/iot-market-forecast-to-2030-connections-by-region-and-vertical`.

Guo, Q., Gu, R., Wang, Z., Zhao, T., Ji, Y., Kong, J., Gour, R., and Jue, J. P. (2019a). Proactive dynamic network slicing with deep learning based short-term traffic prediction for 5G transport network. In *Optical Fiber Communication Conference*, pages W3J–3.

Guo, S., Lin, Y., Feng, N., Song, C., and Wan, H. (2019b). Attention based spatial-temporal graph convolutional networks for traffic flow forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 922–929.

Gupta, O. and Raskar, R. (2018). Distributed learning of deep neural network over multiple agents. *Journal of Network and Computer Applications*, 116:1–8.

Hafi, H., Brik, B., Frangoudis, P. A., Ksentini, A., and Bagaa, M. (2024). Split federated learning for 6g enabled-networks: Requirements, challenges, and future directions. *IEEe Access*, 12:9890–9930.

Han, B., Gopalakrishnan, V., Ji, L., and Lee, S. (2015). Network function virtualization: Challenges and opportunities for innovations. *IEEE communications magazine*, 53:90–97.

Han, D.-J., Bhatti, H. I., Lee, J., and Moon, J. (2021). Accelerating federated learning with split learning on locally generated losses. In *International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML (FL-ICML)*. Poster.

Hedayat, K., Krzanowski, R., Morton, A., Yum, K., and Babiarz, J. (2008). A two-way active measurement protocol (TWAMP)-RFC 5357. *IETF, Standard Specification*.

Hoffman, K. and Padberg, M. (1985). Lp-based combinatorial problem solving. *Annals of Operations Research*, 4:145–194.

Howard, A. G. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hsu, C.-H. and Kremer, U. (1998). IPERF: A framework for automatic construction of performance prediction models. In *Workshop on Profile and Feedback-Directed Compilation (PFDC)*.

Hu, Y. C., Patel, M., Sabella, D., Sprecher, N., and Young, V. (2015). Mobile edge computing—a key technology towards 5G. *ETSI white paper*, 11(11):1–16.

Huin, N., Jaumard, B., and Giroire, F. (2018). Optimal network service chain provisioning. *IEEE/ACM Transactions on Networking*, 26(3):1320–1333.

Hwang, J., Noh, B., Jin, Z., and Yeo, H. (2022). Asymmetric long-term graph multi-attention network for traffic speed prediction. In *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pages 1498–1503.

Hyodo, N., Sato, T., Shinkuma, R., and Oki, E. (2019). Virtual network function placement for service chaining by relaxing visit order and non-loop constraints. *IEEE Access*, 7:165399–165410.

Ickin, S., Larsson, H., Riaz, H., Lan, X., and Kilinc, X. (2022). Decentralized learning and intelligent automation: the key to zero-touch networks? [online]. https://www.ericsson.com/en/blog/2022/3/decentralized-learning-for-zero-touch-networks.

Ilyas, I. F. and Rekatsinas, T. (2022). Machine learning and data cleaning: Which serves the other? *ACM Journal of Data and Information Quality (JDIQ)*, 14(3):1–11.

InterDigital and Research, A. (2021). Environmentally sustainable 5g deployment. https://www.datacenter-forum.com/datacenter-forum/5g-will-prompt-energy-consumption-to-grow-by-staggering-160-in-10-years. Accessed: 2025-05-19.

International Energy Agency (2021). Data centres and data transmission networks. https://www.iea.org/energy-system/buildings/data-centres-and-data-transmission-networks. Accessed: 2025-05-19.

Iqbal, A., Gope, P., and Sikdar, B. (2025). Privacy-preserving collaborative split learning framework for smart grid load forecasting. *IEEE Transactions on Dependable and Secure Computing*.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *IEEE conference on computer vision and pattern recognition*, pages 2704–2713.

Jaumard, B. and Ziazet, J. (2023). 5G E2E network slicing predictable traffic generator. In *International Conference on Network and Service Management (CNSM)*, pages 1–7.

Jeon, J. and Kim, J. (2020). Privacy-sensitive parallel split learning. In *International Conference on Information Networking (ICOIN)*, pages 7–9.

Ji, H., Park, S., Yeo, J., Kim, Y., Lee, J., and Shim, B. (2018). Ultra-reliable and low-latency communications in 5G downlink: Physical layer aspects. *IEEE Wireless Communications*, 25(3):124–130.

Jiang, C., Cheng, X., Gao, H., Zhou, X., and Wan, J. (2019). Toward computation offloading in edge computing: A survey. *IEEE Access*, 7:131543–131558.

Kabir, H. D., Khosravi, A., Mondal, S. K., Rahman, M., Nahavandi, S., and Buyya, R. (2021). Uncertainty-aware decisions in cloud computing: Foundations and future directions. *ACM Computing Surveys (CSUR)*, 54(4):1–30.

Kairouz, P., McMahan, H. B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A. N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al. (2021). Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311.

Kasuluru, V., Blanco, L., Vaca-Rubio, C. J., and Zeydan, E. (2024). On the impact of prb load uncertainty forecasting for sustainable open RAN. In *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–7.

Kasuluru, V., Blanco, L., and Zeydan, E. (2023). On the use of probabilistic forecasting for network analysis in open ran. In *2023 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pages 258–263. IEEE.

Khachiyan, L. G. (1980). Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53–72.

Kim, D., Ko, M., Kim, S., Moon, S., Cheon, K.-Y., Park, S., Kim, Y., Yoon, H., and Choi, Y.-H. (2022). Design and implementation of traffic generation model and spectrum requirement calculator for private 5G network. *IEEE Access*, 10:15978–15993.

Kim, S., Han, Y., and Park, S. (2016). An energy-aware service function chaining and reconfiguration algorithm in NFV. In *IEEE International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 54–59.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Klinkowski, M. (2020). Latency-aware DU/CU placement in convergent packet-based 5G fronthaul transport networks. *Applied Sciences*, 10(21):7429.

Klinkowski, M. (2024). Optimized planning of DU/CU placement and flow routing in 5G packet Xhaul networks. *IEEE Transactions on Network and Service Management*, 21(1):232–248.

Konečnỳ, J., McMahan, H. B., Ramage, D., and Richtárik, P. (2016). Federated optimization: Distributed machine learning for on-device intelligence. *arXiv preprint arXiv:1610.02527*.

Kougioumtzidis, G., Poulkov, V. K., Lazaridis, P. I., and Zaharis, Z. D. (2025). Mobile network traffic prediction using temporal fusion transformer. *IEEE Transactions on Artificial Intelligence*.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S., and Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76.

Kuang, W., Qian, B., Li, Z., Chen, D., Gao, D., Pan, X., Xie, Y., Li, Y., Ding, B., and Zhou, J. (2024). FederatedScope-LLM: A comprehensive package for fine-tuning large language models in federated learning. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 5260–5271.

Lai, F., Zhu, X., Madhyastha, H., and Chowdhury, M. (2021). Oort:efficient federated learning via guided participant selection. In *Symp. on Operating Systems Design and Implementation*, pages 19 – 35.

Land, A. H. and Doig, A. G. (2009). An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, pages 105–132. Springer.

LeCun, Y. and C, C. (2010). Mnist handwritten digit database. Accessed: Jul. 30, 2023.

LeCun, Y. et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lee, J., Seif, M., Cho, J., and Poor, H. V. (2024). Exploring the privacy-energy consumption tradeoff for split federated learning. *IEEE Network*, 38(6):388–395.

Letaief, K. B., Shi, Y., Lu, J., and Lu, J. (2021). Edge artificial intelligence for 6G: Vision, enabling technologies, and applications. *IEEE Journal on Selected Areas in Communications*, 40(1):5–36.

Leyva-Pupo, I. and Cervelló-Pastor, C. (2022). Efficient solutions to the placement and chaining problem of user plane functions in 5G networks. *Journal of Network and Computer Applications*, 197:103269.

Leyva-Pupo, I., Cervelló-Pastor, C., and Llorens-Carrodeguas, A. (2019). Optimal placement of user plane functions in 5G networks. In *Wired/Wireless Internet Communications (WWIC)*, pages 105–117. Springer.

Li, C., Zeng, X., Zhang, M., and Cao, Z. (2022a). PyramidFL: a fine-grained client selection framework for efficient federated learning. In *International ACM Conference on Mobile Computing And Networking (MobiCom)*, page 158–171.

Li, D., Hong, P., Xue, K., and Pei, J. (2019). Virtual network function placement and resource optimization in NFV and edge computing enabled networks. *Computer Networks*, 152:12–24.

Li, H., Assis, K., Vafeas, A., Yan, S., and Simeonidou, D. (2022b). Resilient and energy efficient DU-CU-MEC deployments for service oriented reliable next generation metro access network. *47th WWRF*.

Li, J., Wei, H., Liu, J., and Liu, W. (2024a). FSLEdge:an energy-aware edge intelligence framework based on federated split learning for industrial internet of things. *Expert Systems With Applications*, 255:124564.

Li, M., Soltanolkotabi, M., and Oymak, S. (2020a). Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. In *International conference on artificial intelligence and statistics*, pages 4313–4324. PMLR.

Li, Q., Han, Z., and Wu, X.-M. (2018a). Deeper insights into graph convolutional networks for semi-supervised learning. In *AAAI Conference on Artificial Intelligence*, volume 32.

Li, S., Da Xu, L., and Zhao, S. (2018b). 5G internet of things: A survey. *Journal of Industrial Information Integration*, 10:1–9.

Li, T., Sahu, A. K., Talwalkar, A., and Smith, V. (2020b). Federated learning: Challenges, methods, and future directions. *IEEE signal processing magazine*, 37(3):50–60.

Li, Y., Liang, W., Li, J., Cheng, X., Yu, D., Zomaya, A. Y., and Guo, S. (2023). Energy-aware, device-to-device assisted federated learning in edge computing. *IEEE Transactions on Parallel and Distributed Systems*, 34(7):2138–2154.

Li, Y., Yu, R., Shahabi, C., and Liu, Y. (2017). Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*.

Li, Z., Wu, W., Wu, S., and Wang, W. (2024b). Adaptive split learning over energy-constrained wireless edge networks. *arXiv preprint arXiv:2403.05158*.

Lin, Z., Qu, G., Chen, X., and Huang, K. (2024a). Split learning in 6G edge networks. *IEEE Wireless Communications*.

Lin, Z., Qu, G., Wei, W., Chen, X., and Leung, K. K. (2024b). AdaptSFL: Adaptive split federated learning in resource-constrained edge networks. *arXiv preprint arXiv:2403.13101*.

Liu, H., Ding, S., Wang, S., Zhao, G., and Wang, C. (2022). Multi-objective optimization service function chain placement algorithm based on reinforcement learning. *Journal of Network and Systems Management*, 30(4):58.

Liu, J., Zhao, B., Shao, M., Yang, Q., and Simon, G. (2020). Provisioning optimization for determining and embedding 5G end-to-end information centric network slice. *IEEE Transactions on Network and Service Management*, 18(1):273–285.

Lu, Z., Pan, H., Dai, Y., Si, X., and Zhang, Y. (2024). Federated learning with non-iid data: A survey. *IEEE Internet of Things Journal*.

Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, 53:1007–1023.

Luo, B., Xiao, W., Wang, S., Huang, J., and Tassiulas, L. (2022). Tackling system and statistical heterogeneity for federated learning with adaptive client sampling. In *INFOCOM*, pages 1739–1748.

Ma, Q., Sun, W., Gao, J., Ma, P., and Shi, M. (2023). Spatio-temporal adaptive graph convolutional networks for traffic flow forecasting. *IET Intelligent Transport Systems*, 17(4):691–703.

Maciel, F., de Souza, A. M., Bittencourt, L. F., Villas, L. A., and Braun, T. (2024). Federated learning energy saving through client selection. *Pervasive and Mobile Computing*, 103(101948).

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. (2017). Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR.

Mechtri, M., Ghribi, C., and Zeghlache, D. (2016). A scalable algorithm for the placement of service function chains. *IEEE Transactions on Network and Service Management*, 13(3):533–546.

Merkwirth, C. and Lengauer, T. (2005). Automatic generation of complementary descriptors with molecular graph networks. *Journal of Chemical Information and Modeling*, 45(5):1159–1168.

Michalewicz, Z. and Fogel, D. B. (2013). *How to solve it: modern heuristics*. Springer Science & Business Media.

Mijumbi, R., Serrat, J., Gorricho, J.-L., Bouten, N., De Turck, F., and Boutaba, R. (2015). Network function virtualization: State-of-the-art and research challenges. *IEEE Communications surveys & tutorials*, 18(1):236–262.

Mortazavi, S. and Sousa, E. (2023). Efficient mobile cellular traffic forecasting using spatial-temporal graph attention networks. In *IEEE Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pages 1–6.

Mtawa, Y. A., Haque, A., and Bitar, B. (2019). The mammoth internet: Are we ready? *IEEE Access*, 7:132894–132908.

Murphy, R. L., Srinivasan, B., Rao, V., and Ribeiro, B. (2019). Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations (ICLR)*, pages 1–22.

Nardini, G., Sabella, D., Stea, G., Thakkar, P., and Virdis, A. (2020). Simu5g–an omnet++ library for end-to-end performance evaluation of 5G networks. *IEEE Access*, 8:181176–181191.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. (1978). An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming*, 14:265–294.

O-RAN Alliance (2022). O-RAN Architecture Description v3.0. `https://www.o-ran.org/specifications`. Accessed: 2024-07-29.

Oh, Y., Lee, J., Brinton, C. G., and Jeon, Y.-S. (2023). Communication-efficient split learning via adaptive feature-wise compression. arXiv.

Pang, G., Shen, C., Cao, L., and Hengel, A. V. D. (2021). Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*, 54(2):1–38.

Park, J., Samarakoon, S., Bennis, M., and Debbah, M. (2019). Wireless network intelligence at the edge. *Proceedings of the IEEE*, 107(11):2204–2239.

Parvez, I., Rahmati, A., Guvenc, I., Sarwat, A. I., and Dai, H. (2018). A survey on low latency towards 5G: RAN, core network and caching solutions. *IEEE Communications Surveys & Tutorials*, 20(4):3098–3130.

Patriciello, N., Lagen, S., Bojovic, B., and Giupponi, L. (2019). An E2E simulator for 5G NR networks. *Simulation Modelling Practice and Theory*, 96:101933.

Pei, J., Hong, P., Pan, M., Liu, J., and Zhou, J. (2019). Optimal VNF placement via deep reinforcement learning in SDN/NFV-enabled networks. *IEEE Journal on Selected Areas in Communications*, 38(2):263–278.

Pei, J., Hong, P., Xue, K., and Li, D. (2018). Efficiently embedding service function chains with dynamic virtual network function placement in geo-distributed cloud system. *IEEE Transactions on Parallel and Distributed Systems*, 30(10):2179–2192.

Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2018). Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, 30(2):339–360.

Postel, J. (1981). Internet Control Message Protocol. Technical report, IETF Network Working Group, RFC-792.

Qi, D., Shen, S., and Wang, G. (2019). Towards an efficient VNF placement in network function virtualization. *Computer Communications*, 138:81–89.

Raca, D., Leahy, D., Sreenan, C. J., and Quinlan, J. J. (2020). Beyond throughput, the next generation: a 5G dataset with channel and context metrics. In *ACM multimedia systems conference*, pages 303–308.

Rahman, S., Mun, H., Lee, H., Lee, Y., Tornatore, M., and Mukherjee, B. (2018). Insights from analysis of video streaming data to improve resource management. In *IEEE International Conference on Cloud Networking (CloudNet)*, pages 1–3.

Randal, A. (2020). The ideal versus the real: Revisiting the history of virtual machines and containers. *ACM Comput. Surv.*, 53(1).

Rao, A., Tärneberg, W., Fitzgerald, E., Corneo, L., Zavodovski, A., Rai, O., Johansson, S., Berggren, V., Riaz, H., Kilinc, C., et al. (2022). Prediction and exposure of delays from a base station perspective in 5G and beyond networks. In *ACM SIGCOMM Workshop on 5G and Beyond Network Measurements, Modeling, and Use Cases*, pages 8–14.

Rossem, S. V., Tavernier, W., Colle, D., Pickavet, M., and Demeester, P. (2019). Profile-based resource allocation for virtualized network functions. *IEEE Transactions on Network and Service Management*, 16(4):1374–1388.

Samani, F. (2023). Data traces for efficient learning on high-dimensional operational data. `https://github.com/foroughsh/KTH-traces`. Accessed: Jul. 30, 2023.

Samikwa, E., Di Maio, A., and Braun, T. (2022). ARES: Adaptive resource-aware split learning for Internet of things. *Computer Networks*, 218:109380.

Savoia, M., Prezioso, E., Mele, V., and Piccialli, F. (2024). Eco-FL: enhancing federated learning sustainability in edge computing through energy-efficient client selection. *Computer Communications*, 225:156 – 170.

Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2008). The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80.

Schuß, M., Boano, C. A., Weber, M., and Römer, K. (2017). A competition to push the dependability of low-power wireless protocols to the edge. In *European Conference on Wireless Sensor Networks (EWSN)*, pages 54–65.

Shullary, M. H., Abdellatif, A. A., and Massoudn, Y. (2022). Energy-efficient active federated learning on non-IID data. In *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1–4.

Singh, A. K. and Nguyen, K. K. (2022). Joint selection of local trainers and resource allocation for federated learning in open ran intelligent controllers. In *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1874–1879.

Song, C., Wu, J., Xian, K., Huang, J., and Lu, L. (2024). Spatio-temporal graph learning: Traffic flow prediction of mobile edge computing in 5g/6g vehicular networks. *Computer Networks*, 252:110676.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Steinwart, I. and Christmann, A. (2011). Estimating conditional quantiles with the help of the pinball loss. *arXiv preprint arXiv:1102.2101*.

Stephanie, V., Khalil, I., and Atiquzzaman, M. (2023). Digital twin enabled asynchronous SplitFed learning in E-healthcare systems. *IEEE Journal on Selected Areas in Communications*, 41(11):3650 – 3661.

Stich, S. U., Cordonnier, J.-B., and Jaggi, M. (2018). Sparsified SGD with memory. *Advances in neural information processing systems*, 31.

Stuart, C., Filipovic, Z., and Ramiro, J. (2023). How is AI boosting network performance? Accessed: Sep. 12, 2023.

Sun, J., Zhang, Y., Liu, F., Wang, H., Xu, X., and Li, Y. (2022). A survey on the placement of virtual network functions. *Journal of Network and Computer Applications*, 202:103361.

Sun, P., Lan, J., Li, J., Guo, Z., and Hu, Y. (2020). Combining deep reinforcement learning with graph neural networks for optimal VNF placement. *IEEE Communications Letters*, 25(1):176–180.

Talbi, E.-G. (2009). *Metaheuristics: from design to implementation*. John Wiley & Sons.

Taleb, T., Samdanis, K., Mada, B., Flinck, H., Dutta, S., and Sabella, D. (2017). On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681.

Tang, H., Zhou, D., and Chen, D. (2018). Dynamic network function instance scaling based on traffic forecasting and VNF placement in operator data centers. *IEEE Transactions on Parallel and Distributed Systems*, 30(3):530–543.

Tarng, P.-Y., Chen, K.-T., and Huang, P. (2008). An analysis of WoW players' game hours. In *ACM SIGCOMM Workshop on Network and System Support for Games, (NETGAMES)*, pages 1 – 7, Worcester, Massachusetts, USA.

Tavallaee, M., Bagheri, E., Lu, W., and Ghorbani, A. A. (2009). A detailed analysis of the KDD CUP 99 data set. In *IEEE sympos. on computational intelligence for security and defense applications*, pages 1–6.

Thapa, C., Arachchige, P. C. M., Camtepe, S., and Sun, L. (2022). Splitfed: When federated learning meets split learning. In *AAAI Conference on Artificial Intelligence*, volume 36, pages 8485–8493.

Tran, N. P., Delgado, O., Jaumard, B., and Bishay, F. (2023). ML KPI prediction in 5G and B5G networks. In *Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*, pages 502–507.

Turina, V., Zhang, Z., Esposito, F., and Matta, I. (2021). Federated or split? a performance and privacy analysis of hybrid split and federated learning architectures. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 250–260.

Vaca-Rubio, C. J., Kasuluru, V., Zeydan, E., Blanco, L., Pereira, R., Caus, M., and Dev, K. (2025). Probabilistic forecasting for network resource analysis in integrated terrestrial and non-terrestrial networks. *IEEE Communications Standards Magazine*.

Varga, A. and Hornig, R. (2008). An overview of the OMNeT++ simulation environment. In *International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, pages 1–10.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, 30.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations (ICLR)*, pages 1–12.

Vepakomma, P., Gupta, O., Swedish, T., and Raskar, R. (2018). Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*.

Vishwanath, A., Hinton, K., Ayre, R. W., and Tucker, R. S. (2014). Modeling energy consumption in high-capacity routers and switches. *IEEE Journal on Selected Areas in Communications*, 32:1524–1532.

Walia, J., Hämmäinena, H., Kilkki, K., and Yrjölä, S. (2019). 5G network slicing strategies for a smart factory. *Computers in Industry*, 111:108 – 120.

Wang, J., Qi, H., Rawat, A. S., Reddi, S., Waghmare, S., Yu, F. X., and Joshi, G. (2022a). Fedlite: A scalable approach for federated learning on resource-constrained clients. *arXiv preprint arXiv:2201.11865*.

Wang, J., Tang, J., Xu, Z., Wang, Y., Xue, G., Zhang, X., and Yang, D. (2017). Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach. In *IEEE conference on computer communications (INFOCOM)*, pages 1–9.

Wang, T., Zu, J., Hu, G., and Peng, D. (2020). Adaptive service function chain scheduling in mobile edge computing via deep reinforcement learning. *IEEE Access*, 8:164922–164935.

Wang, X., Wang, Z., Yang, K., Song, Z., Bian, C., Feng, J., and Deng, C. (2024). A survey on deep learning for cellular traffic prediction. *Intelligent Computing*, 3:0054.

Wang, X., Zhou, Z., Xiao, F., Xing, K., Yang, Z., Liu, Y., and Peng, C. (2018). Spatio-temporal analysis and prediction of cellular traffic in metropolis. *IEEE Transactions on Mobile Computing*, 18(9):2190–2202.

Wang, Z., Hu, J., Min, G., Zhao, Z., Chang, Z., and Wang, Z. (2022b). Spatial-temporal cellular traffic prediction for 5g and beyond: A graph neural networks-based approach. *IEEE Transactions on Industrial Informatics*, 19(4):5722–5731.

Wu, H. and Wang, P. (2021). Fast-convergent federated learning with adaptive weighting. *IEEE Transactions on Cognitive Communications and Networking*, 7(4):1078–1088.

Wu, Q., Chen, X., Zhou, Z., Chen, L., and Zhang, J. (2021). Deep reinforcement learning with spatio-temporal traffic forecasting for data-driven base station sleep control. *IEEE/ACM transactions on networking*, 29(2):935–948.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2020a). A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24.

Wu, Z., Pan, S., Long, G., Jiang, J., Chang, X., and Zhang, C. (2020b). Connecting the dots: Multivariate time series forecasting with graph neural networks. In *ACM SIGKDD international conference on knowledge discovery & data mining*, pages 753–763.

Xiao, Y., Zhang, J., and Ji, Y. (2021). Energy-efficient DU-CU deployment and lightpath provisioning for service-oriented 5g metro access/aggregation networks. *Journal of Lightwave Technology*, 39(17):5347–5361.

Xiong, L., Su, L., Wang, X., and Pan, C. (2024). Dynamic adaptive graph convolutional transformer with broad learning system for multi-dimensional chaotic time series prediction. *Applied Soft Computing*, 157:111516.

Xu, C., Qu, Y., Xiang, Y., and Gao, L. (2023). Asynchronous federated learning on heterogeneous devices: A survey. *Computer Science Review*, 50:100595.

Xu, X., Cao, H., Geng, Q., Liu, X., Dai, F., and Wang, C. (2022). Dynamic resource provisioning for workflow scheduling under uncertainty in edge computing environment. *Concurrency and Computation: Practice and Experience*, 34(14):e5674.

Yan, S., Xiong, Y., and Lin, D. (2018). Spatial temporal graph convolutional networks for skeleton-based action recognition. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Yan, W. (2018). Machine learning for enabling active measurements in IoT environments. Technical report, Royal Institute of Technology (KTH), Sweden.

Yang, Z., Chen, M., Wong, K.-K., Poor, H. V., and Cui, S. (2022). Federated learning for 6G: Applications, challenges, and opportunities. *Engineering*, 8:33–41.

Yanggratoke, R. et al. (2018). A service-agnostic method for predicting service metrics in real time. *International Journal of Network Management*, 28(2):e1991.

Yeom, H., Ko, H., and Pack, S. (2023). Traffic-efficient split computing mechanism for Internet of Things. In *Int'l Conf. on Information and Communication Technology Convergence (ICTC)*, pages 495–496.

Yousaf, F. Z., Bredel, M., Schaller, S., and Schneider, F. (2017). Nfv and sdn—key technology enablers for 5g networks. *IEEE Journal on Selected Areas in Communications*, 35(11):2468–2478.

Yu, B., Yin, H., and Zhu, Z. (2017). Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting. *arXiv preprint arXiv:1709.04875*.

Yu, H., Musumeci, F., Zhang, J., Xiao, Y., Tornatore, M., and Ji, Y. (2019). DU/CU placement for C-RAN over optical metro-aggregation networks. In *International IFIP Conference on Optical Network Design and Modeling*, pages 82–93. Springer.

Yu, L., Li, M., Jin, W., Guo, Y., Wang, Q., Yan, F., and Li, P. (2020). STEP: A spatio-temporal fine-granular user traffic prediction system for cellular networks. *IEEE Transactions on Mobile Computing*, 20(12):3453–3466.

Yuan, B., Ge, S., and Xing, W. (2020). A federated learning framework for healthcare IoT devices. *arXiv preprint arXiv:2005.05083*.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. *Advances in Neural Information Processing Systems*, 30.

Zha, D., Bhat, Z. P., Lai, K.-H., Yang, F., Jiang, Z., Zhong, S., and Hu, X. (2025). Data-centric artificial intelligence: A survey. *ACM Computing Surveys*, 57(5):1–42.

Zhang, C., Patras, P., and Haddadi, H. (2019a). Deep learning in mobile and wireless networking: A survey. *IEEE Communications surveys & tutorials*, 21(3):2224–2287.

Zhang, H., Liu, N., Chu, X., Long, K., Aghvami, A.-H., and Leung, V. C. (2017). Network slicing based 5G and future mobile networks: Mobility, resource management, and challenges. *IEEE communications magazine*, 55(8):138–145.

Zhang, J., Chen, B., and Ye, Y. (2005). A multiexchange local search algorithm for the capacitated facility location problem. *Mathematics of Operations Research*, 30(26):389 – 403.

Zhang, J., Li, Z., Li, B., Xu, J., Wu, S., Ding, S., and Wu, C. (2022). Federated learning with label distribution skew via logits calibration. In *Int'l Conf. on Machine Learning*, pages 26311–26329. PMLR.

Zhang, K., Zhu, Y., Leng, S., He, Y., Maharjan, S., and Zhang, Y. (2019b). Deep learning empowered task offloading for mobile edge computing in urban informatics. *IEEE Internet of Things Journal*, 6:7635–7647.

Zhang, Z., Xiao, Y., Ma, Z., Xiao, M., Ding, Z., Lei, X., Karagiannidis, G. K., and Fan, P. (2019c). 6G wireless networks: Vision, requirements, architecture, and key technologies. *IEEE Vehicular Technology Magazine*, 14(3):28–41.

Zhao, J., Feng, Y., Chang, X., and Liu, C. (2022). Energy-efficient client selection in federated learning with heterogeneous data on edge. *Peer-to-Peer Networking and Applications*, page 1139–1151.

Zhao, L., Song, Y., Zhang, C., Liu, Y., Wang, P., Lin, T., Deng, M., and Li, H. (2019). T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems*, 21(9):3848–3858.

Zhao, N., Wu, A., Pei, Y., Liang, Y.-C., and Niyato, D. (2021a). Spatial-temporal aggregation graph convolution network for efficient mobile cellular traffic prediction. *Communications Letters*, 26(3):587–591.

Zhao, S., Liu, Z., Lin, J., Zhu, J.-Y., and Han, S. (2020). Differentiable augmentation for data-efficient gan training. *Advances in Neural Information Processing Systems*, 33:7559–7570.

Zhao, T., Liu, Y., Neves, L., Woodford, O., Jiang, M., and Shah, N. (2021b). Data augmentation for graph neural networks. In *AAAI Conference on Artificial Intelligence*, volume 35, pages 11015–11023.

Zhao, Z., Feng, C., Hong, W., Jiang, J., Jia, C., Quek, T. Q., and Peng, M. (2021c). Federated learning with non-IID data in wireless networks. *IEEE Transactions on Wireless communications*, 21(3):1927–1942.

Zheng, F., Chen, C., Lyu, L., and Yao, B. (2023). Reducing communication for split learning by randomized top-k sparsification. In *International Joint Conference on Artificial Intelligence*, pages 4665–4673.

Zhou, Z., Li, Y., Ren, X., and Yang, S. (2022). Towards efficient and stable k-asynchronous federated learning with unbounded stale gradients on non-IID data. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):3291–3305.

Ziazet, J., Jaumard, B., Duong, H., Khoshabi, P., and Janulewicz, E. (2022a). A dynamic traffic generator for elastic 5G network slicing. In *IEEE International Symposium on Measurements & Networking (M&N)*, pages 1–6.

Ziazet, J., Jaumard, B., Larabi, A., and Huin, N. (2023). Placement of logical functionalities in 5G/B5G networks. In *IEEE Future Networks World Forum (FNFW)*, pages 1 – 7, Baltimore (MD) US.

Ziazet, J. M., Boudreau, C., Jaumard, B., and Duong, H. (2022b). Addressing routenet scalability through input and output design. *ITU Journal on Future and Evolving Technologies*.

Zorello, L. M. M., Sodano, M., Troia, S., and Maier, G. (2022). Power-efficient baseband-function placement in latency-constrained 5G metro access. *IEEE Transactions on Green Communications and Networking*, 6(3):1683–1696.