SAPIENZA
UNIVERSITÀ DI ROMA

Anomaly Detection in Mobile Edge Computing Infrastructures using Application Logs: an Adaptive Federated Learning and Finite Time Consensus-based Approach

Facoltà di Ingegneria dell'informazione, informatica e statistica
Master's Degree in Control Engineering

Stefano Felli
ID number 1896877

Advisor
Prof. Emanuele De Santis

Co-Advisor
Dr. Danilo Menegatti

Academic Year 2023/2024

Thesis defended on 30-th October 2024
in front of a Board of Examiners composed by:

Prof. Francesco Delli Priscoli (chairman)
Prof. Irene Amerini
Prof. Alessandro Giuseppi
Prof. Emanuele De Santis
Prof. Francesco Liberati
Prof. Mattia Mattioni
Prof. Flavio Pepe
Prof. Antonio Pietrabissa

Anomaly Detection in Mobile Edge Computing Infrastructures using Application Logs: an Adaptive Federated Learning and Finite Time Consensus-based Approach
Master Thesis. Sapienza University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: enofel2000@gmail.com

You miss 100% of the shots you don't take
-Michael Scott

Contents

Abstract

Anomaly detection systems play a vital role in securing Critical Infrastructures (CIs), which are large-scale distributed systems including Mobile Edge Computing (MEC) environments. These infrastructures often rely on computer-based controllers that continuously generate logs to monitor their operations. This Thesis proposes two complementary log anomaly detection frameworks based on Federated Learning (FL), which ensures that sensitive data remains localized while enabling collaborative learning across distributed agents. The first algorithm, named AdaLightLog, enhances the traditional centralized FL paradigm by incorporating adaptive loss functions at local servers and employing a weighted averaging process during global model aggregation, which accounts for the varying quality of local models across different sites. In contrast, the second presented DecAdaLightLog approach implements a decentralized FL framework, where the learning process is distributed across the participating agents using finite-time consensus protocol. In this way there is no need for a central coordinator server that could represents a single point of failure targeted by cyber-physical attacks. The effectiveness of both approaches is evaluated against state-of-the-art methods, demonstrating improved performance in terms of accuracy, precision, and recall compared to the standard FL implementation (FedAvg). Additionally, a comprehensive comparison of different metrics used for adaptive loss functions and dynamic weighting in both centralized and decentralized settings is provided, highlighting their impact on overall system performance in the context of MEC.

Sommario

Anomaly detection svolge un ruolo fondamentale nella sicurezza delle Infrastrutture Critiche (CIs), come ad esempio gli ambienti Mobile Edge Computing (MEC). Questi sistemi sono distribuiti su larga scala e spesso si basano su controllori computerizzati che generano continuamente log per monitorare le loro operazioni. Questa Tesi propone due framework complementari per il rilevamento delle anomalie nei log basati su Federated Learning (FL), assicurando che i dati sensibili rimangano localizzati, consentendo al contempo un apprendimento collaborativo tra agenti distribuiti. Il primo algoritmo, chiamato AdaLightLog, migliora il tradizionale paradigma FL centralizzato incorporando funzioni di perdita adattive nei server locali e aggregando i pesi del modello globale secondo una media ponderata. Il secondo approccio presentato, DecAdaLightLog, implementa un framework FL decentralizzato, in cui il processo di addestramento dei clienti che vi partecipano è reso possibile grazie all'applicazione del protocollo di consenso a tempo finito, senza la necessità di un server centrale, che potrebbe rappresentare un elemento di vulnerabilità per eventuali attacchi cyber-informatici. L'efficacia di entrambi gli approcci è valutata rispetto ai metodi tradizionali, dimostrando un miglioramento delle prestazioni in termini di accuratezza, precisione e richiamo rispetto all'implementazione FL standard (FedAvg). Inoltre, viene fornito un confronto completo tra le diverse metriche utilizzate per le funzioni di perdita adattive e la ponderazione dinamica sia in contesti centralizzati che decentralizzati, evidenziando il loro impatto sulle prestazioni complessive del sistema nel contesto MEC.

Chapter 1

Introduction

Anomaly detection is an essential task for ensuring correct functionality of Critical Infrastructures (CIs). These systems are essential for providing vital economic and social functions, such as energy production, transportation, healthcare, and financial systems [8]. The smooth operation of CIs is fundamental to the well-being of society and the economy.

These infrastructures traditionally heavily depend on Industrial Control Systems (ICSs), responsible to the control of physical processes and management of operations in many sectors like energy, manufacturing, and transportation. However, the advent of modern technologies, particularly the Internet of Things (IoT) [50] and Mobile Edge Computing (MEC), is transforming how CIs operate.

MEC represents a more recent technological advancement that brings computational resources closer to the network's edge, next to end-users and IoT devices. This infrastructure consists of strategically deployed edge nodes and IoT devices, each fulfilling distinct roles. Edge nodes are powerful computing units positioned near data sources to handle tasks such as data processing, storage, and networking locally. IoT devices, such as smartphones, laptops, sensors and cameras, generate data and interact with their relative edge node. While traditional Cloud Computing [33] relies on a centralized architecture where raw data must travel to and from geographically distant data centers before being processed, MEC operates in a distributed manner. In fact edge nodes are positioned next to IoT devices, that provide them raw data for processing with significantly reduced latency. Thus, due to its distributed architecture, MEC is an important tool for high-bandwidth, low-latency and privacy preserving applications that are critical for services like autonomous vehicles, smart healthcare, and industrial automation [51]. Figure 1.1 highlights the difference between the two approaches.

Still, though, its distributed nature, characterized by interoperability between heterogeneous devices, makes MEC infrastructures more vulnerable than others to cyber-attacks on physical environments. These attacks can increase latency, disrupt service availability, and even cause system-wide failures. For instance, a Distributed Denial-of-Service (DDoS) attack targeting edge nodes could lead to delays that make IoT devices unable to perform critical tasks in real time. In healthcare, such delays might result in medical devices failing to function as required, potentially endangering patients. In industrial automation, increased latency could lead to

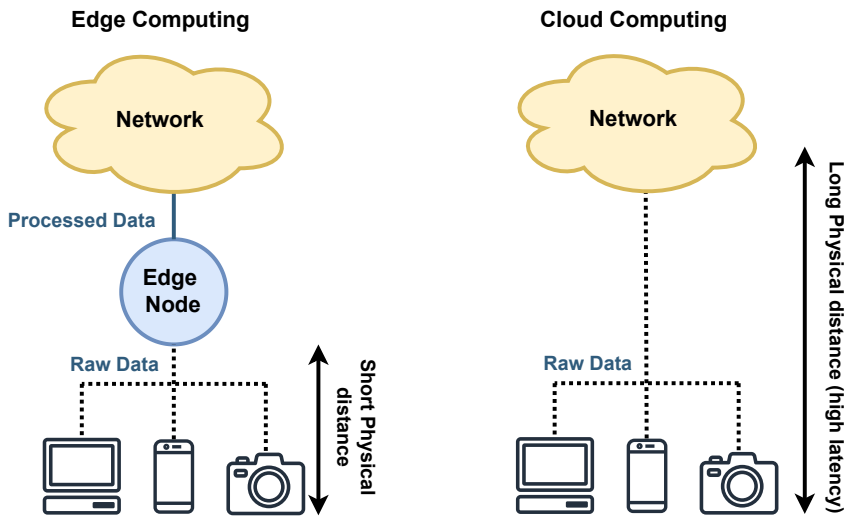


Figure 1.1 Edge Computing Infrastructure (left) compared with traditional Cloud Computing framework (right)

malfunctions in robotic systems, causing safety risks. Furthermore, in augmented and virtual reality applications, which depend heavily on MEC to process high volumes of data with minimal delay, any disruption can severely degrade the user experience or compromise the safety of the environment. These examples illustrate the importance of ensuring that MEC infrastructures remain secure and resilient against cyber threats to maintain the reliability of various time-sensitive services.

An important anomaly detection task for the maintenance of the security and functionality of MEC infrastructures is *log anomaly detection*. In fact, system logs are the most relevant data generated by IoT devices. Once they are collected and processed by MEC edge nodes, log data provide a detailed record of events and operations that help in identifying anomalies that might indicate incoming attacks or system failures. Early detection of such anomalies is particularly important, as it allows for immediate responses to potential threats before they evolve into larger issues. The increasing frequency and sophistication of attacks necessitate the development of accurate and effective anomaly detection systems, often based on Machine Learning (ML) models, to preventing the above mentioned issues. In MEC environments, where logs are often expressed in human-readable natural language, advanced parsing and embedding techniques are required to transform them into formats suitable for analysis by ML models.

This Thesis introduces two novel methodologies for distributed log anomaly detection tailored to the specific challenges of MEC infrastructures, leveraging the Federated Learning (FL) paradigm to maintain data privacy across distributed edge nodes. FL is particularly well-suited for environments like MEC, where the sensitivity of the data processed at the edge necessitates robust privacy measures. By enabling collaborative model training without the need to share raw data, FL ensures that privacy is preserved while still enabling effective anomaly detection. Both methods introduce advanced log parsing and embedding strategies inspired by

the approaches discussed in [9] and [59], ensuring that logs are efficiently processed and embedded into formats suitable for ML models.

The first methodology, AdaLightLog, is a federated and adaptive extension of the algorithm presented in [59]. It adopts a centralized FL strategy, building upon the principles introduced by the AdaFed algorithm [12]. Specifically, it enhances the traditional process by incorporating adaptive loss functions at the local server level and a performance-based weighted averaging procedure at the central server, both of which are key features of AdaFed. This strategy ensures that the global model accurately reflects the quality of individual models trained at various edge nodes, thereby improving the overall robustness and accuracy of the system.

The second approach, DecAdaLightLog, extends the FL paradigm to a decentralized setting. In this configuration, model training, consensus-building, and the application of adaptive loss functions occur locally at each edge node. This decentralized approach is particularly advantageous in scenarios where maintaining a central server is impractical or undesirable, such as in highly distributed MEC networks where avoiding a single point of failure is crucial. This complete decentralization is achieved in this work thanks to the innovative idea of applying a finite-time consensus protocol for multi-agent systems. Unlike the centralized method, DecAdaLightLog does not utilize weighted model averaging, primarily because this process requires a common test dataset accessible to all nodes, which is not feasible in a decentralized environment.

This Master's Thesis introduces several innovations in the field. The main contributions are resumed as follows:

- Application of a centralized FL methodology, named AdaLightLog, to the log anomaly detection problem in MEC and IoT infrastructures, that preserves the privacy of the various clients participating in the environment;
- application of a finite-time consensus-based FL framework, named DecAdaLightLog, to the log anomaly detection problem, eliminating the need for a central central server that represents a single point of failure targeted by cyber-physical attacks, then ensuring complete decentralization of the learning process;
- application of Physics-Informed Neural Networks to automatically learn suitable Lyapunov functions for a specific task – such as for achieving consensus within finite time – replacing the need for complex analytical derivations;
- introduction of a performance-based weighted parameter averaging procedure at the central server in AdaLightLog to prioritize more reliable client contributions in learning, mitigating the effects of potential malicious attacks on individual clients (characterized by poor performance);
- improvement of the training process through adaptive loss functions linked to validation metrics for each client in both centralized and decentralized settings, that gives more importance to miss-classified data samples;
- metric-based comparative analysis of training performance across different FL models and with respect to the traditional FedAvg algorithm.

This work has been partially supported by the EU in the scope of the NANCY project, which has received funding from the Smart Networks and Services Joint Undertaking (SNS JU) under the European Union's Horizon Europe research and innovation programme under Grant Agreement No 101096456, and the FSE REACT-EU, PON Ricerca e Innovazione 2014-2020, programme.

A first part of the Thesis was published in the proceedings of the 19th International Conference on Critical Information Infrastructures Security (CRITIS2024) – submitted on 31/05/2024 and discussed on 18/09/2024 – and was a finalist for the prestigious Young CRITIS Award (YCA). Given the significant innovative contribution and the surprising results of the second algorithm, it is planned to submit the complete work for further publication in a scientific journal.

Chapter 2

State of Art

In recent years, the vast amount of logs generated daily by CIs has necessitated the development of various automatic anomaly detection techniques. However, when these logs contain sensitive information, sharing them for training purposes poses significant privacy challenges. This issue has led to a growing interest in privacy-preserving approaches to anomaly detection, especially within distributed environments like MEC.

Anomaly detection task has been extensively studied due to its importance in maintaining situational awareness and ensuring the security and operational integrity of CIs. Early works, such as those by [55] and [24], relied on purely statistical methods to detect anomalies by leveraging the statistical properties of attack patterns. While effective for certain applications, these methods often struggle to generalize to new types of attacks and more complex infrastructures.

As Machine Learning methodologies gained traction, they offered a promising alternative by requiring less domain-specific knowledge and showing better adaptability to emerging threats. For instance, [2, 19, 23] explored the use of Support Vector Machines (SVMs) and Bayesian Networks for anomaly classification in computer systems. However, these approaches were soon surpassed by advances in Deep Learning, which demonstrated superior performance in detecting anomalies, particularly in complex environments such as computer networks, as seen in [17, 45].

When narrowing down to the specific task of *log anomaly detection*, which is particularly relevant for MEC infrastructures, the process typically involves two main phases: transforming raw, unstructured logs into a structured format for analysis, and then using this processed data to identify abnormal patterns that may indicate system issues or security breaches.

For the log processing phase, this work employs the same methodology as LightLog [59], which involves parsing unstructured logs into structured log keys using tools like Spell [37], Drain [18] and IPLoM [34, 35]. These parsers convert raw log messages into sequences of events, which are then embedded into dense vector representations using techniques such as Word2Vec [21]. This method efficiently prepares the log data for subsequent analysis.

Building on the log processing approach used in LightLog, this research focuses on enhancing the anomaly detection phase. Various classification methods have been explored in the past to detect anomalies in processed logs. One of the earliest

approaches, presented by Xu et al. [54], utilized Principal Component Analysis (PCA) to mine console logs for large-scale system issues. This method effectively identifies patterns in high-dimensional data but may struggle with the complexity of modern systems. Another notable approach is the invariant mining-based method introduced by Lou et al. [32]. This technique involves mining program workflows from interleaved traces to identify invariants, which are conditions that hold true during normal operation. Anomalies are detected when these invariants are violated, making this method particularly useful for detecting errors in complex software systems. Additionally, workflow-based methods have been developed to identify execution anomalies in program logic flows. Yu et al. [56] proposed CloudSeer, a framework that monitors cloud infrastructure workflows by analyzing interleaved logs. CloudSeer focuses on detecting deviations in the expected execution order of workflows, helping to uncover anomalies in the program logic. Even though these methods have been successful in specific scenarios, they often fall short when it comes to adapting to diverse and dynamic environments, particularly in online settings where immediate detection is critical.

To address these challenges, more advanced DL techniques have been developed. DeepLog [9] was introduced as a more adaptive approach to anomaly detection in logs. DeepLog learns the patterns in system-generated log sequences by using as its core a model inspired by the Long Short-Term Memory (LSTM) network, a kind of Recurrent Neural Network (RNN). Though it trains on normal log event sequences, DeepLog can predict in real time what the next event in any given sequence would be and flag any significant deviations from that prediction as anomalous. As a result, DeepLog is especially adept at performing anomaly detection in real time within highly dynamic environments. Building on the ideas behind DeepLog, LightLog [59] was developed as a more efficient approach to centralized log anomaly detection. LightLog focuses on reducing the computational and storage overhead associated with log processing and classification. It achieves this by simplifying the parsing and embedding processes, making it a lightweight yet accurate solution for detecting anomalies in logs. However, like DeepLog, LightLog operates within a centralized framework, which can pose challenges in scenarios where data privacy and decentralized processing are essential.

Federated Learning provides a decentralized approach to model training, particularly advantageous for privacy-sensitive environments like MEC. Recent research has focused on enhancing FL to handle non-IID data, reduce communication overhead, and improve robustness against stragglers. Various works, such as [57] and [29], have introduced methods to address these challenges, making FL increasingly suitable for distributed anomaly detection tasks in MEC infrastructures.

This research applies the FL paradigm for anomaly detection, a concept initially explored by Preuveneers et al. [46] in training an intrusion detection model using the CIC-IDS2017 dataset. Building on this foundation, [52] introduced FedSam, a FedAvg-like algorithm, which addresses the challenges posed by stragglers through the introduction of a data sampling strategy and a distributed min-max scaling approach to manage data heterogeneity. The issue of data distribution is further explored in [40], where clients are grouped based on the statistical characteristics of their local datasets, with FedAvg being applied to these clusters. Privacy concerns are tackled in [53] and [20]; the former integrates a hierarchical structure with

blockchain technology for server-side validation of model updates, while the latter employs the SSL/TLS protocol.

The first objective of this work is to develop a centralized anomaly detection framework, named AdaLightLog, that begins with application-level log data while ensuring the privacy of the collected user data through the AdaFed algorithm [12]. This approach contrasts with methods presented in [27] and [15], which rely on FedAvg. The proposed framework enhances robustness against malicious attacks by incorporating a model testing phase before server updates, rather than merely adding Gaussian noise to the model parameters at the client level. Additionally, the presented method optimizes the computational demands of the learning process by introducing an adaptive loss function, differing from [15]'s reliance on the lottery ticket hypothesis [10], which could expose clients to increased risks of attacks.

Consensus theory is one of the most basic notions in distributed systems. A group of nodes or agents agrees on a single value of data or decision against faults or divulgation of conflicting information. In other words, consensus ensures consistency and reliability across an unmanaged network with no central authority emanating decisions. Some of consensus algorithms, such as Paxos [25] and Raft [43], have lately been studied and applied to a wide range of distributed systems, from blockchain technologies [39] to large-scale cloud infrastructures [7]. In these systems, consensus ensures that all participating nodes converge on a common state despite the presence of network partitions, node failures, or malicious actors. In particular, consensus theory becomes relevant for FL because it provides the means by which a set of decentralized clients can update global models collaboratively without the need to trust a centralized server [28]. However, existing methods rely on infinite-time consensus protocols, such as the one proposed by Giuseppi et al. [13], where agents reach agreement, but the time required for convergence may be indefinite. This can limit the efficiency of the learning process, particularly in time-sensitive applications. To overcome such a limitation, the second objective of this thesis is to propose a novel FL approach, called DecAdaLightLog, that leverages finite-time consensus mechanisms. Building on the theoretical results from Zheng et al. [58], which achieve finite-time consensus in continuous-time systems, this work adapts these principles to the FL context. While Zheng et al. do not directly address FL, their methods provide a foundation for ensuring rapid convergence of decentralized models, significantly improving the efficiency and robustness of FL frameworks. In addition to the continuous-time approach, recent work by Cacace et al. [5] introduces a distributed protocol that achieves finite-time consensus for discrete-time systems. Their results demonstrate that rapid convergence can be attained even in discrete temporal frameworks, where the number of time steps required for convergence can be arbitrarily defined. In this work, both continuous-time and discrete-time consensus techniques have been employed in the development of the FL algorithm which is guaranteed to converge in bounded time.

Physics-Informed Neural Networks (PINNs) are one of the advanced techniques at the interface of ML with physics. It has a unique capability wherein the NN will get trained not only on data but also on governing physical laws of the system. PINNs, proposed by Raissi et al. [47], have demonstrated their efficiency for solving a large class of forward-inverse problems, especially nonlinear partial differential equations. PINNs directly embed knowledge of physics into the loss function and are

able to learn complex dynamics that are analytically hard or impossible to model.

In this framework, PINNs are employed to discover a suitable Lyapunov function that meets the criteria for finite-time consensus as outlined by Zheng et al. This innovative use of PINNs leverages the power of DL to automate the process of stability analysis, thereby enhancing both the robustness and efficiency of the proposed decentralized FL framework. By ensuring that model convergence occurs within a bounded time, this method is particularly well-suited for real-time applications in dynamic MEC environments. Overall, the proposed framework offers a scalable and secure alternative to centralized FL, aligning with the privacy-preserving principles critical to modern anomaly detection systems [11].

Chapter 3

Federated Learning

Federated Learning is a decentralized approach in Machine Learning that enables a group of devices or organizations to jointly train a model over their local data. This method is very applicable when data privacy and security, coupled with regulatory compliance, is a matter of utmost concern. This effort at model training need not necessarily be centrally done using sensitive data.

Before going into the details of Federated Learning, it is important to briefly go over some of the basics of Deep Learning (DL) and Artificial Neural Networks (ANNs), since most of the times they are part of the core of the FL process. DL is a subset of ML that concerns the modeling and interpretation of patterns in data with the help of a Neural Network. An ANN, also known as Feedforward Neural Network (FNN), Multi-layer Perceptron (MLP) or simply Neural Network (NN), can be understood as a complex function parameterized by weights W, that maps input data x to output predictions $\hat{y}$.

$$\hat{y} = \mathcal{F}(x; W)$$

It consists of layers of interconnected nodes, called neurons, that process the input data through mathematical operations. Each layer takes the output of the previous one, applies affine transformations by multiplying its input with weights, followed by non-linear activation functions that introduce non-linearity into the model, allowing it to learn complex patterns, and passes the result forward through the network to produce the final prediction $\hat{y}$.

The learning process in NNs, illustrated in Figure 3.1, involves the adjustment of its internal parameters W, which determine the strength of connections between neurons, and are updated in an iterative way during training. The goal of training is, for any given input-output pair in the dataset $D = \{(x, y)\}$, to minimize the error in the network's predictions, represented by a *loss function* $\ell(D, W)$ such that the model output $\hat{y}$ well approximates the ground-truth value y. Loss functions are in fact used to quantify the quality of the output produced by the NN given a generic input sample with respect its actual output. This procedure is accomplished through optimization techniques, such as *Stochastic Gradient Descent* (SGD) [14] or *Adaptive Moment Estimation* (Adam) [22], which systematically adjusts the weights W in the direction of the negative gradient of the loss function $\nabla\ell(D, W)$, computed at each epoch as a combination of the gradients of the activation functions in each layer,

influencing how errors propagate backward through the network. Specifically for SGD, during a generic iteration i-th, named epoch, weights are updated according to the formula:

$$W(i+1) = W(i) - \eta \cdot \nabla \ell(D, W(i)) \tag{3.1}$$

Where η is a constant value, named *learning rate*, that controls the step size of the parameter update.

Progressive adjustment of the weights brings convergence to a set of optimal weights W^{opt} such that $\nabla \ell(D, W^{\text{opt}}) = 0$, that best generalize new unseen input data x' with unknown output. This occurs provided the learning rate is small enough to prevent significant oscillations around W^{opt}, ensuring stable convergence.

$$y' = \mathcal{F}(x'; W^{\text{opt}}), \quad W^{\text{opt}} = \text{argmin}\{\ell(D, W)\}$$

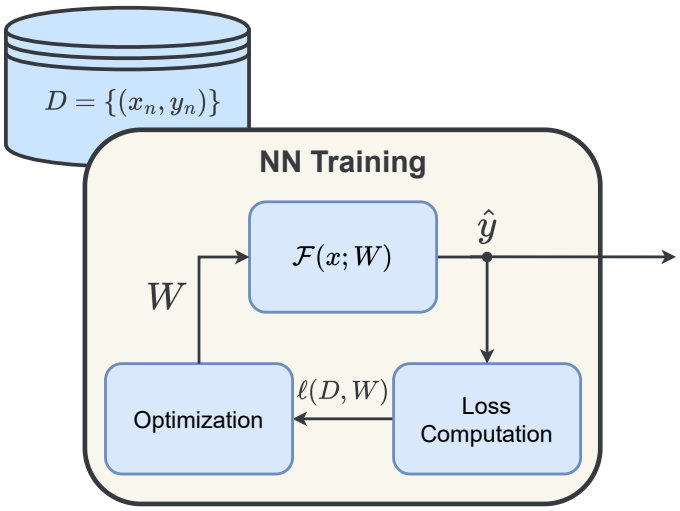


Figure 3.1 Neural Networks training algorithm

In distributed environments, traditional centralized ML approaches requires the collection of all data items to a central server for processing. However, this kind of centralized approach is unworkable in many cases when applied to MEC infrastructures, considering several factors such as bandwidth limitation, high latency, and violation of data privacy. In fact, large volumes of data are transferred from distributed edge devices to the central server; this leads to network resource overload, resultant network congestion, and delays. Besides, transferring sensitive data centrally causes significant privacy concerns because of increased risks of data breach and unauthorized access.

Addressing these challenges, FL empowers the training process to take place right where the data is being acquired, directly on local devices, without any need for centralized data aggregation. In FL, only model updates, such as changes in the weights of NNs, are shared with a central server or among participants, which reduces the need for data transfers drastically. This decentralized approach not only minimizes congestion over the network but also enhances privacy and security because sensitive information would remain on the local devices. Thus, FL turns

out to be particularly fitted for MEC environments where maintaining data privacy, reducing latency, and efficient radio resource exploitation are of major importance.

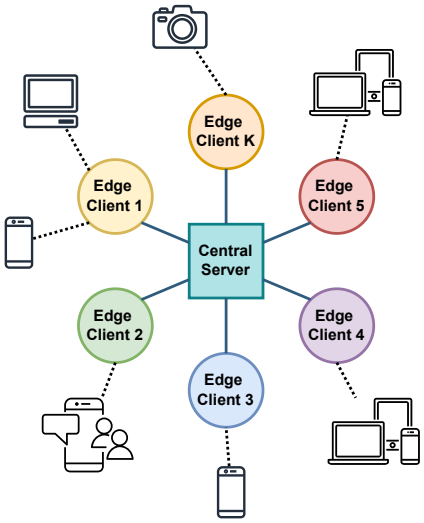


Figure 3.2 Centralized Federated Learning Architecture

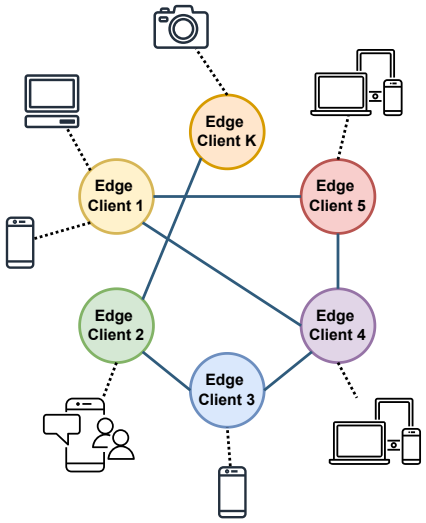


Figure 3.3 Decentralized Federated Learning Architecture

FL can be broadly classified into two primary approaches. The centralized framework involves a central server coordinating the entire learning process. Every participating node k, such as a mobile device or edge server, trains a NN model $\mathcal{F}_k(D_k; W_k)$ on its local data D_k, sending updates in the model parameters to the central server; it aggregates the received parameters and computes an average to provide a global model that gets redistributed to the nodes. This approach is relatively easier to implement, and it performs well when the central server is trusted and can efficiently deal with the aggregation process. Figure 3.2 presents the centralized FL architecture applied in a MEC infrastructure, where all the edge nodes upload their model parameters to a central server for aggregation.

In contrast, decentralized FL doesn't rely on any central server. In the latter setting, as illustrated in Figure 3.3, clients communicate directly with their neighbours, broadcasting model updates and jointly converging to an agreement about the global model. While such peer-to-peer communication is more challenging and therefore requires an effective synchronization mechanism possibly combined with a consensus-building protocol, it has considerable advantages for settings where the risk of single-point failure needs to be avoided or where no centralized coordination is possible. Decentralized systems contribute to resilience and robustness, since they do not depend on one server that may become a point of congestion or a target of attacks.

A prime example and probably the most popular centralized FL algorithm, Federated Averaging (FedAvg), was proposed by McMahan et al. in 2017 [36]. It considers a straightforward yet effective way to combine knowledge gained from multiple local models into a single global model. FedAvg is a centralized FL

algorithm that works as follows: Each client locally trains a DL model, such as a NN, on its local data, and thereafter sends the updated model parameters to the central server. The updated parameters received by the server from all the clients are then combined by weighted averaging wherein the weights are taken according to the rule depending upon the amount of data used during training by the client. This aggregated global model is then redistributed to each client, and this process is iterated until convergence. The convergence of FedAvg was first established in the context of next-word prediction [16]. Later, the authors of [30] presented an formal proof of its $\mathcal{O}\left(\frac{1}{\tau}\right)$ convergence rate, where τ denotes the number of local training iterations at the client level. More advanced techniques such as Nesterov momentum [4] have further improved the convergence rate in convex settings, which makes variants of FedAvg very effective in distributed learning.

Since its proposal, there have been many uses of FedAvg across domains, which demonstrate the robustness of the paradigm of FL. In the context of IoT and MEC, FL has been developed to carry out multiple models for predictive maintenance, detecting anomalies, and resource allocation. With FL, these models can be trained on a network of edge devices without having to centralize sensitive data, thus keeping privacy intact while reducing latency [41]. In the medical domain, FL empowered the model training on patient data contributed from different hospitals, while sensitive information on patients remained in the hospitals, and the model learned globally from an assorted set [1].

Yet, at the same time, FL also possesses several drawbacks despite its many advantages. One of the most basic problems is the communication overhead because updates are very frequent among clients and the server –or among peers in the case of decentralized FL. This can easily become a serious bottleneck in MEC, where usually for each edge node there is not much bandwidth available, and latency is a crucial issue. Compared to traditional centralized approaches, which send all data once, FL requires iterative communication that may strain network resources and delay the process of training. Besides, FL contains complex architecture that binds the participating devices in a well-coordinated manner. In fact, maintaining all the clients in synchronization and effectively contributing to the model training process is not easy; it may be incredibly tricky in an environment with a variation of devices with computational power and network connectivity.

Chapter 4

Centralized Federated Learning Anomaly detection Strategy

This chapter outlines the strategy developed for anomaly detection in distributed log data using Federated Learning. The presented method is designed to operate effectively in a centralized environment.

In this work, the log anomaly detection problem is formulated as a *binary classification* task, where the goal is to determine whether each log instance represents normal or abnormal activity. Unlike traditional anomaly detection methods, which typically involve collecting raw logs in a centralized manner for uniform pre-processing, the proposed approach handles scenarios where logs remain stored on the edge devices. Each client is responsible for parsing, sampling, embedding and train its own data, which was previously collected by its associated IoT devices.

4.1 Log Processing Workflow

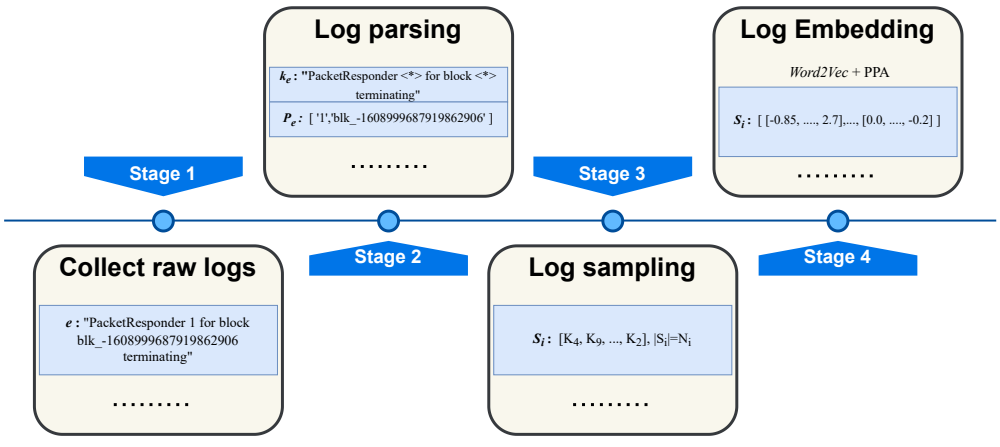


Figure 4.1 Chosen log processing framework

The pre-processing stage of logs, is aimed at transforming raw logs into meaningful representations for the training of the learning model. Various strategies proposed

in [59], which enable real-time log mapping with minimal computational resource consumption, are employed, processing logs in sequential phases as shown in Figure 4.1.

Considering their entries as textual data containing various fields, such as timestamps and details regarding system processing, it is crucial to eliminate unnecessary information to isolate particular values. After each edge node collects the log data from its associated IoT devices (Stage 1), it locally processes logs firstly by parsing them (Stage 2). Log parsing methods such like SPELL [37], DRAIN [18] and IPLoM [34, 35] convert each raw log entry e into a structured template. This process identifies a frequently repeated portion of the log, known as log key k_e, and removes the varying parts, defined as event parameters p_e.

Once each entry e is parsed, the sampling phase takes place (Stage 3), with the log keys k_e being sampled into execution sequences S_i based on the nature of the log data, each one with a different number N_i of log keys. This is achieved using session window methods, such as those applied to Hadoop Distributed File System (HDFS) logs [54] and OpenStack (OS) logs [9], which can be grouped by *block id* field (HDFS) or *instance id* field (OS). For logs that do not include session identifiers, like Blue Gene/L (BGL) logs [31], grouping can be done based on timestamps.

Finally, the the log embedding phase (Stage 4) concludes the pre-processing workflow. In the context of log processing, Word Embedding refers to the process of mapping textual data (in this case, discrete log keys) into continuous numerical vectors. This transformation is crucial because it enables Machine Learning models, particularly Neural Networks, to process and understand the data – originally expressed in human-readable natural language – in a mathematical form that preserves its underlying structure.

This transformation is performed by many different types of word embedders, all of which have their own way of accounting for the relationships between words. A popular approach is using the WORD2VEC model [21], a NN that learns vector representations of words in high-dimensional space based on their proximity to each other within the data. Words that are found to be in similar areas in this space get placed closer to each other, representing their semantic proximity. Another method is GLOVE (Global Vectors for Word Representation) [44], which aims to learn global "word-word co-occurrence" statistics between words by considering overall global judgments across the full text of a data corpus. This enables GLOVE to produce embeddings that capture both local context and global structure. Another embedding technique is FASTTEXT [3], which extends WORD2VEC to sub-word units. This way, FASTTEXT is able to handle rare words, typos, or morphological rich languages because it embodies semantics at a finer level.

For the considered workflow, the structured sequences of log keys S_i are encoded using the WORD2VEC model. The choice of WORD2VEC is motivated by its effectiveness in environments where the vocabulary is relatively well-defined and where the occurrence of rare words or misspellings is minimal, as is the case with logs. For this kind of data, the log keys are typically consistent and repeated, making the granular subword information captured by FASTTEXT less critical. Furthermore, WORD2VEC provides a good balance between computational efficiency and the ability to capture meaningful semantic relationships, which is essential

given the large volume of log data that needs to be processed. Each log key k_e is projected into an $\mathcal{M}$-dimensional vector $V_e^{\mathcal{M}}$, called *word embedding*, where $\mathcal{M}$ is sufficiently large to ensure a comprehensive representation of the log key's semantic information. Consequently, the processed log sequences are composed of numerical vectors $V_1^{\mathcal{M}}, ..., V_{N_i}^{\mathcal{M}}$. Such word embeddings can be computationally expensive to handle, particularly in tasks involving extensive training data or large vocabularies.

To overcome these computational challenges and enhance efficiency, dimensionality reduction techniques such as the Post-Processing Algorithm (PPA) are commonly employed [48]. PPA's primary objective is to decrease the dimensionality of Word2Vec embeddings while retaining as much semantic information as possible. This reduction in dimensionality substantially alleviates the computational burden associated with processing and storing the embeddings, making them more suitable for downstream tasks like ML and Natural Language Processing (NLP).

The PPA, proposed in [48] and reported in Algorithm 1, utilizes techniques such as Principal Component Analysis (PCA) to achieve dimensionality reduction. PCA identifies the principal components of the high-dimensional embedding space and projects the embeddings onto a lower-dimensional subspace $n < \mathcal{M}$ while preserving as much variance as possible. Additionally, PPA removes d dominating eigenvectors from the dimensionality-reduced embeddings. These dominating eigenvectors, which contribute the most to the variance in the data, are removed to potentially eliminate noise or irrelevant information from the embeddings. The specific number of eigenvectors to remove is typically determined empirically or through cross-validation, depending on the characteristics of the data and the specific application.

Algorithm 1 Post Processing Algorithm (PPA)

Inputs: $V^{\mathcal{M}}$: $\mathcal{M}$-dimensional vectors, d: hyper-parameter
Output: V^n: $n < \mathcal{M}$-dimensional vectors

1: $V^{PCA} \leftarrow \text{PCA}(V^{\mathcal{M}})$
2: $V^{PCA} \leftarrow V^{PCA} - Avg(V^{PCA})$
3: $V^n \leftarrow \text{PCA}(V^{PCA})$
4: **for** v **in** V^n **do**
5: Remove d dominating eigenvectors from v
6: **end for**
7: **return** V^n

4.2 Temporal Convolutional Network Architecture for Local Training

In this work, as also proposed in [9, 59], the training process for embedded data utilizes a Temporal Convolutional Network (TCN) firstly introduced in [26]. TCNs are particular Convolutional Neural Networks (CNNs) which proved to be effective for the log anomaly detection problem [59] due to their known capabilities in recognition of temporal patterns like the sequences of log keys. Given as input to the NN, a log

sequence classified as anomalous is characterized by an irregular pattern over time, with log keys fluctuating between unusual and normal values, normal otherwise.

To well understand TCNs, it's helpful to first consider the structure of a CNN, a class of NNs commonly used for processing data with a grid-like structure, such as videos, images or sequences. It is a composition of both convolutional and classic NN layers. Each convolutional layer performs a sequence of operations. The first one is commonly called *convolution operation*: it is an affine transformation of the input that involves applying a *filter* of a fixed size, also known as kernel, to the input data. Such filter slides across the input, producing an output, named *feature map*, where each element is the weighted sum of the input values covered by the kernel. To this output is then applied a nonlinear *activation function* Φ that introduces a nonlinearity in the affine transformation. This step is fundamental to learn more complex relationships in data with respect to a linear model. The resulting output is often reshaped by a *pooling layer* that reduces its dimension. There are two main classes of convolutional layers, depending on the nature of the convolution operation.

- **Causal convolutions:** In *causal convolutions*, the output at any given time step depends only on the current and past inputs, ensuring that future data does not influence the prediction for the current time step. From a mathematical perspective, given an input sequence $x = \{x_{[1]}, ..., x_{[q]}\}$, where each input $x_{[t]} \in \mathbb{R}^n$ is the n-dimensional word embedding that encodes the log key produced at the generic time $t = 1, \ldots, q$, a causal convolution operation with a kernel size F produces the output:

$$y_{[t]} = \sum_{f=0}^{F-1} w_f \, x_{[t-f]}, \tag{4.1}$$

  where w_f are the weights of the convolutional filter. This equation shows that the output $y_{[t]}$ at time t is a weighted sum of the current embedding and the previous $F - 1$ inputs.

- **Dilated convolutions:** *Dilated convolutions* extend the idea of causal ones by introducing a *dilation rate* r, which essentially controls the spacing between the input values that are considered. Instead of taking F consecutive embeddings, the dilated convolution aims at capturing temporal dependencies at different scales, allowing the network to look at inputs further apart in the sequence without increasing the size of the filter. The feature map resulting from a dilated convolution operation is:

$$y_{[t]} = \sum_{f=0}^{F-1} w_f \, x_{[t-rf]}. \tag{4.2}$$

  Here, the input values $x_{[t-rf]}$ are spaced by the dilation rate r, enabling the network to capture patterns that occur over longer intervals of time without losing finer details. To further clarify, when the dilation rate $r = 1$, a dilated convolutional layer behaves exactly like a causal one as the network processes consecutive inputs without skipping any values, making the two types of convolutions functionally identical in this specific case.

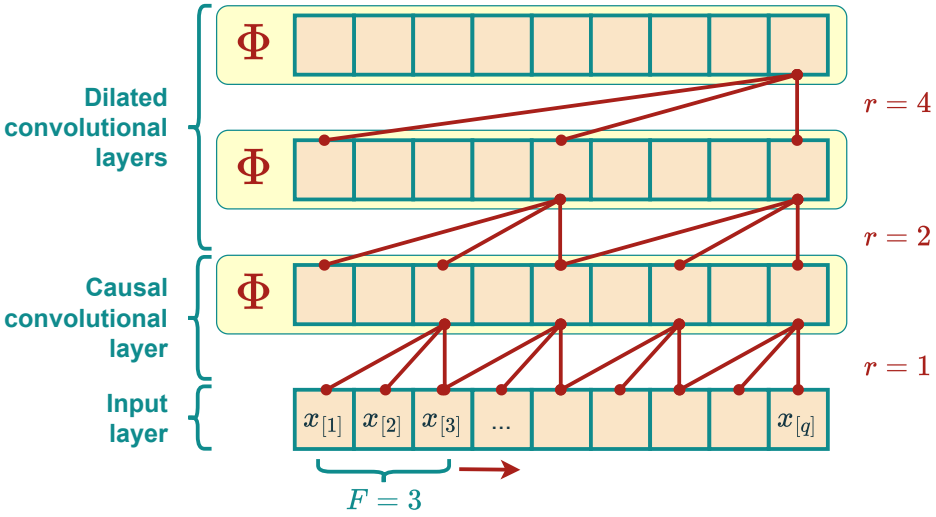


Figure 4.2 Illustration of a stacked sequence of convolutional layers, comprising one causal layer and two dilated layers. Each layer applies the convolution operation followed by a nonlinear activation function, while the pooling stage is neglected

Figure 4.2 illustrates a general sequence of convolutional layers, where different dilation rates are used to capture temporal dependencies at multiple scales. The example shown has a kernel size $F = 3$ and dilation rates $r = 1, 2$ and 4, respectively. These values are chosen for illustrative purposes to demonstrate how dilated convolutions can expand the receptive field of the network, allowing it to capture both short-term and long-term patterns within the input sequence.

The main component of a TCN is the so called *residual block*, which is critical for capturing temporal dependencies in sequential data. A residual block is made up of a set of causal and dilated convolutional layers, with the dilated ones sharing same dilation rate. Each nonlinearly transformed feature map is not reshaped with pooling layers. A *residual connection* (shortcut path) bypasses the convolutional layers and directly adds the input of the block to its output. This design ensures the activation functions gradients acting on each layer remain meaningful and helps to mitigate the *vanishing gradient problem*, a situation that occurs during training DNNs when such structures stop learning entirely as as the loss function's gradient scales down to zero: this happens when the activation functions in hidden layers enter their flat regions, causing their gradients to vanish. As a result, the computation of the loss gradient is affected. The residual connections help by allowing gradients to flow more easily through the network, ensuring that the model can learn efficiently even in deep architectures, thereby improving the efficiency and effectiveness of training. An example of residual block is shown in Figure 4.3, composed by a dilated and a causal layer.

The overall TCN architecture is constructed by stacking multiple residual blocks, with the dilation rate of the convolutions increasing exponentially across them. This stacked design allows the network to capture dependencies over a wide range of temporal scales, making it particularly well-suited for tasks such as anomaly detection,

where patterns can vary significantly in their temporal context. The output of these residual blocks is typically handled by *global pooling layers*, which aggregate the temporal features extracted by the network, summarizing the information across the entire sequence. This pooled output is then processed by *fully connected layers*, providing the final output for binary classification of the log sequence.

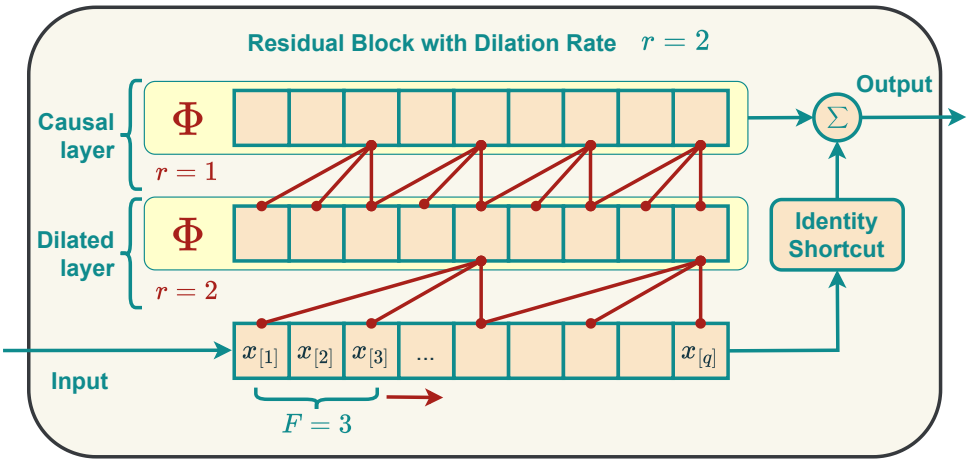


Figure 4.3 Structure of a residual block with a dilated layer (dilation rate $r = 2$) followed by a causal one

4.3 Proposed Centralized Anomaly Detection Algorithm

The proposed AdaLightLog framework introduces a relevant innovation over the LightLog algorithm [59] on which it is based. In fact the approach in analysis envisages a federation of K local servers in communication with a coordinating one, denoted as G, as shown in Figure 4.4.

Such a scenario is quite customary in CIs, especially within MEC environments, where such infrastructures rely on edge devices capable of storing and processing data, such as local servers, connected to the outside environment via some secure connection. With respect to the coordinating server, it can be either one among the local ones, or may be located in the cloud and can even belong to a different party. The inherent nature of the FL paradigm allows different stakeholders to share the same algorithm without compromising data privacy, as no raw logs or embeddings are exchanged among the federation members. Instead, as previously mentioned, each local server independently handles data pre-processing, resulting in unique embeddings that closely reflect real-world applications.

A key innovation distinguishing this approach from other log-based FL implementations [27, 15] is the incorporation of a *weighted averaging* mechanism and an *adaptive loss function*, both derived from the AdaFed algorithm [12]. These features significantly enhance the learning process.

The weighted averaging procedure allows the central server G to prioritize more reliable client contributions, enhancing the robustness of the global model by mitigating the effects of potential malicious attacks on individual clients. By

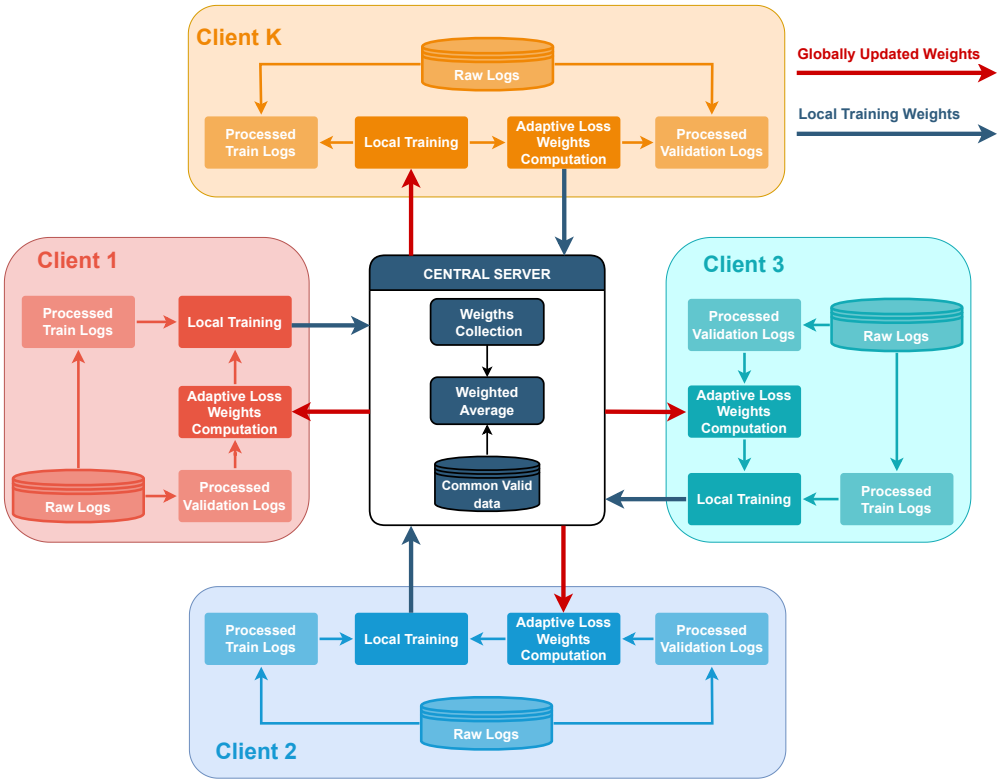


Figure 4.4 AdaLightLog framework

evaluating the performance of client updates through a metric-driven validation process at the central server G, the system can identify anomalies or inconsistencies in the model parameters submitted by clients. If a client's contribution is suspected to be compromised, possibly due to an attack, the central server can exclude that client's updates from the parameters aggregation process.

Simultaneously, the adaptive loss function facilitates faster convergence by dynamically adjusting in response to each client's performance, giving more importance to miss-classified data samples. The evaluation is conducted on the client's local validation data using the aggregated parameters after the weighted averaging process. Each client's loss function is specifically adapted to its unique validation dataset, resulting in distinct loss function weights across clients that ensure the efficient evolution of the global model, even in the presence of data heterogeneity, This dual capability not only strengthens the overall security of the FL framework but also ensures that the resulting global model remains reliable and robust, even in adversarial environments.

Going deeply in details, following a pre-processing phase where each client generates local training and validation datasets, the AdaLightLog training procedure is executed as follows: dealing with a binary classification problem, where each structured sequence of log keys can be classified as either Normal (class 0) or Anomalous (class 1), during the c-th *communication round*, where $c = 1, \ldots, E$, each client k, where $k = 1, \ldots, K$, trains for τ epochs its TCN model $\mathcal{F}_k^{\text{TCN}}(D_k; W_g^c)$ based on

the local training dataset $D_k = \{(S_1^k, y_1^k) \ldots, (S_{|D_k|}^k, y_{|D_k|}^k)\}$, where y_j^k is the actual classification of the log sequence S_j^k, and parameterized by the current weights W_g^c received from the central server G. The local training aims to find, at each local epoch $i = 1, \ldots, \tau$, parameters $W_k^c(i)$ by applying an optimization algorithm to minimize a weighted *categorical cross-entropy* loss function $\ell_k(D_k, c, W)$, defined as follows:

$$\ell_k(D_k, c, W) = -\frac{1}{|D_k|} \sum_{j=1}^{|D_k|} \left[w_{0,k}^c \cdot y_j^k \log(\hat{y}_j^k) + w_{1,k}^c \cdot (1 - y_j^k) \log(1 - \hat{y}_j^k) \right], \quad (4.3)$$

where $\hat{y}_j^k \in \hat{Y}^k = \mathcal{F}_k^{\text{TCN}}(D_k, W)$ is the model's output representing the probability that the log sequence S_j^k is Anomalous (class 1), and $w_{0,k}^c$ and $w_{1,k}^c$ indicate the above mentioned weights associated to the two classes. Initially set to default values, these parameters are locally updated at the end of each c-th communication round, based on a performance metric m_k^{c+1} that reflects the effectiveness of the global model W_g^{c+1} on the client's own validation set:

$$w_{0,k}^{c+1} = \frac{1}{m_k^{c+1} + \varepsilon}, \quad (4.4)$$

$$w_{1,k}^{c+1} = \frac{1}{(1 - m_k^{c+1}) + \varepsilon}, \quad (4.5)$$

where $\varepsilon > 0$ is an arbitrarily small value to prevent numerical problems.

After local training, the client's updated weights, W_k^c, are sent back to the central server G, which evaluates their performance on a common dedicated dataset D_g assumed to be representative of the overall ML problem. The server assigns a performance score s_k^c to each client's model. The global model parameters, W_g^{c+1}, are then updated by performing a weighted average of the collected parameters:

$$W_g^{c+1} = \frac{\sum_{k=1}^K W_k^c \cdot s_k^c}{\sum_{k=1}^K s_k^c}. \quad (4.6)$$

The principle behind equation (4.6) is to allow clients with higher scores to guide the learning process, enabling less effective clients to benefit from their insights. This mechanism helps mitigate the impact of clients with low-quality data or those subjected to malicious attacks. In the former case, the client can improve by leveraging the experience of others, while in the latter, the edge device can be excluded from the learning process without detrimental effects.

The complete process of AdaLightLog FL training is reported in Algorithm 2.

4.4 Problems in Centralized FL

The AdaLightLog framework, while effective for log anomaly detection in MEC environments, faces several challenges due to its centralized nature. The biggest issue is the reliance of the system on a single server acting as the central coordinating point, creating a single *point of failure*. Any disruption, from server downtime to

network issues, and even targeted cyber-attacks, can pause the learning process entirely, making the system less resilient and more vulnerable to failures. This centralized dependency is particularly problematic in MEC environments, where continuous operation is crucial.

Another challenge is the lack of fault tolerance. In a centralized FL system, if the central server becomes overloaded or fails, the learning process stops entirely, affecting the overall performance and reliability of the system. This lack of redundancy makes centralized FL less suitable for highly dynamic and distributed MEC environments where system robustness is critical.

In addition to technical challenges, there are also organizational and policy-related limitations. For compliance reasons, many companies and organizations prohibit the storage of sensitive or proprietary data on central servers. In those scenarios, using a central server as the key coordinator for federated learning is not just impractical but may also violate laws and internal security policies. This also complicates the implementation of centralized FL in environments that demand high standards for privacy, security, and data sovereignty.

These unresolved challenges highlight the limitations of the centralized FL approach and underscore the need for a more resilient, scalable, and secure solution: a consensus-based decentralized FL model, which will be discussed in the next chapter.

Algorithm 2 AdaLightLog: centralized FL Anomaly Detection Strategy

AdaLightLog:
Initialize $w_{0,k}^1$ and $w_{1,k}^1$
for each communication round $c = 1, ..., E$ **do**
    for each client $k = 1, ..., K$ **do**
        $W_k^c \leftarrow$ LocalTraining
        Propagate model weights W_k^c to the global server G
    end for
    $W_g^{c+1} \leftarrow$ WeightedAveraging
    Propagate W_g^{c+1} to the clients
    for each client $k = 1, ..., K$ **do**
        Compute the metric m_k^{c+1} by evaluating W_g^{c+1} on k-th validation dataset
        Compute adaptive loss parameters $w_{0,k}^{c+1}$ and $w_{1,k}^{c+1}$ as in (4.4) and (4.5)
    end for
end for

LocalTraining:
$W_k^c(0) \leftarrow W_g^c$
for each local epoch $i = 1, ..., \tau$ **do**
    Compute predictions $y^k = \mathcal{F}_k^{\text{TCN}}(D_k; W_k^c(i-1))$
    Compute $\ell_k(D_k, c, W_k^c(i-1))$ with adaptive loss weights $w_{0,k}^c$ and $w_{1,k}^c$
    Minimize weights: $W_k^c(i) \leftarrow W_k^c(i-1) - \eta \cdot \nabla \ell_k(D_k, c, W_k^c(i-1))$
end for
Return $W_k^c(\tau)$

WeightedAveraging:
for each client $k = 1, ..., K$ **do**
    Evaluate the score s_k^c over W_k^c on common validation dataset D_g
end for
Perform weighted model average W_g^{c+1} based on the obtained scores as in (4.6)
Return W_g^{c+1}

Chapter 5

Consensus-based FL Anomaly Detection Strategy

The focus of this chapter is on the presentation of DecAdaLightLog, a groundbreaking decentralized adaptation of the AdaLightLog algorithm. This methodology introduces a significant innovation by applying finite-time consensus theory to Federated Learning for log anomaly detection, an area where such integration has not been explored previously. This approach is pivotal for environments characterized by a high degree of node autonomy and variable network conditions, such as MEC infrastructures, effectively addressing the challenges faced by centralized FL. The coordination and learning process are distributed across multiple nodes. Instead of depending on a central server, the consensus mechanism allows each edge device to contribute to the parameters averaging, enhancing resilience against server downtimes, network failures, and cyber-attacks. This distributed nature ensures that the learning process can effectively work even if some nodes for any issue are compromised or offline, significantly improving system robustness.

The mitigation of the central server dependency seen in AdaLightLog also enhances fault tolerance by distributing computational and communication tasks across the network. In DecAdaLightLog framework, if a node fails or becomes overloaded, other clients can compensate, maintaining uninterrupted log anomaly detection. This distributed approach aligns with the dynamic nature of MEC infrastructures, where continuous operation is essential.

5.1 Finite-time Consensus Protocol

To understand the foundation of DecAdaLightLog, it is crucial to first explain the concept of *consensus protocol* within dynamical multi-agent systems (MAS). Consensus protocol is a fundamental mechanism that drives the dynamics of multiple agents of a system in a reliable, efficient and secure way to a common behavior, known as *consensus*, necessary for coordinated actions in decentralized frameworks.

To better understand the formulation of the consensus protocol, it is considered a framework composed by K agents representing the edge devices of a MEC infrastructure. Each client i is modeled as a dynamical system, with an internal state $x_i(t) \in \mathbb{R}^{\mu}$ that, for each instant t, evolves according to its dynamics

$f_i(x_i(t), u_i(x_i(t)))$, dependent by the state itself and the control protocol $u_i(x_i(t))$. By collecting the states of all agents, and omitting explicit time dependence to simplify the notation, there can be defined the state and the relative control protocol matrices $x, u(x) \in \mathbb{R}^{K\mu}$:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix}, \quad u(x) = \begin{bmatrix} u_1(x) \\ u_2(x) \\ \vdots \\ u_K(x) \end{bmatrix} \tag{5.1}$$

The evolution of x follows the dynamical MAS:

$$\begin{cases} \dot{x} = \mathbf{f}\left(x, u(x)\right) \\ x(0) = x_0 \\ \mathbf{f}(0, u(0)) = 0 \end{cases} \tag{5.2}$$

where $\mathbf{f} : \mathbb{R}^{K\mu} \times \mathbb{R}^{K\mu} \to \mathbb{R}^{K\mu}$ is a nonlinear function that governs the evolution of x based on the current states and time, and $\dot{x} \in \mathbb{R}^{K\mu}$ represents the time evolution of x.

Since in a MEC environment the communication between edge devices is typically bidirectional, the topology among such K systems is modeled by an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{x_1, x_2, \ldots, x_K\}$ is the set of nodes corresponding to the agents' states, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges representing the communication links between them. The relationship between any two agents $\{x_i, x_j\} \in \mathcal{E}$ is captured by the adjacency matrix $A \in \mathbb{R}^{K\times K}$, where each entry a_{ij} is a positive value if there is a communication link from agent i to agent j, and zero otherwise. Since $\mathcal{G}$ is undirected, then $\{x_i, x_j\} \in \mathcal{E} \implies \{x_j, x_i\} \in \mathcal{E}$, which implies $a_{ij} = a_{ji}$.

To further describe the communication topology, it can be defined the *degree matrix* $D = \text{diag}(d_1, d_2, \ldots, d_K)$, where each diagonal element $d_i = \sum_{j=1}^{K} a_{ij}$ represents the number of direct connections, called degree, that agent i has with other agents. Using these matrices, the *Laplacian matrix* $L \in \mathbb{R}^{K\times K}$ associated to $\mathcal{G}$ can be defined. It is a fundamental construct in the analysis of consensus protocols for MAS and is given by $L = D - A$. It is a is symmetric ($L = L^T$) and positive semi-definite matrix, meaning that all its eigenvalues λ_i are real and non-negative, ordered as $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_K$. Such a matrix gives information about the connectivity of $\mathcal{G}$: the second smallest eigenvalue of L, λ_2, also known as the *Fiedler eigenvalue*, is greater than zero if and only if the graph $\mathcal{G}$ is connected. The magnitude of λ_2 reflects the connectivity strength of the graph and influences the speed of convergence in consensus algorithms.

The objective of consensus protocol for the defined $\mathcal{G}$ can be formulated mathematically as ensuring that all agents' states converge to a common consensus value $\bar{x}$, defined as the average of the initial states of all agents:

$$\begin{cases} \lim\limits_{t\to\infty} x_i = \bar{x} \quad \forall i = 1, \ldots, K \\ \bar{x} = \dfrac{1}{K} \displaystyle\sum_{i=1}^{K} x_{i,0} \in \mathbb{R}, \quad \forall x_{i,0} \in x_0 \end{cases} \tag{5.3}$$

Finite-time consensus further extends this concept by guaranteeing that all agents reach consensus $\bar{x}$ within a finite time $T \in (0, +\infty)$. This characteristic is particularly vital in MEC infrastructures, where the rapid exchange of information and timely decision-making are essential in such critical and dynamic scenarios in order to maintain the required quality of service. The finite-time consensus is formulated as:

$$\begin{cases} \lim\limits_{t \to T} x_i = \bar{x} \\ x_i(\mathbf{t}) = \bar{x} \quad \forall \mathbf{t} \geq T \end{cases} \tag{5.4}$$

5.1.1 Classical Lyapunov Stability

The Lyapunov stability criterion plays a central role in the study of stability of the dynamical system (5.2). According to this criterion, a system is *locally asymptotically stable* (LAS) at an equilibrium point if there exists a scalar function $V : \mathbb{R}^{K\mu} \to \mathbb{R}_{\geq 0}$, called a *Lyapunov function*, which is continuously differentiable and such that for any solution x the following conditions hold:

1. $V(x)$ is positive definite:

$$V(x) > 0 \quad \forall x \neq 0, \quad \text{and} \quad V(0) = 0.$$

2. The Lie derivative of V along the system's trajectories, denoted as $\dot{V}(x)$, is strictly negative except at the origin:

$$\dot{V}(x) = \nabla V(x) \cdot \dot{x} < 0, \quad \forall x \neq 0.$$

To guarantee *global asymptotic stability* (GAS) at an equilibrium point, an additional condition must be satisfied:

3. $V(x)$ is radially unbounded:

$$V(x) \to \infty \quad \text{as} \quad \|x\| \to \infty.$$

5.1.2 Lyapunov-like Conditions for Finite-time Consensus

Consider the specific MAS of integrators characterized by the following protocols:

$$\begin{cases} \dot{x}_i = u_i(x_i), \\ u_i(x_i) = -\textsc{Sat}_{\Delta_1}\left(k_1 \cdot v_i^{\alpha}(x_i)\right) - k_2 \cdot \textsc{Tanh}\left(v_i(x_i)\right), \\ v_i(x_i) = \sum\limits_{j \in \mathcal{N}_i} (x_i - x_j), \end{cases} \tag{5.5}$$

where v_i represents the standard local control term ensuring agreement among the states of integrator agents, $\mathcal{N}_i = \{x_j \in \mathcal{V} | (x_j, x_i) \in \mathcal{E}\}$ denotes the set of neighbors of agent i ($i = 1, \ldots, K$). $k_1 > 0$ and $k_2 \in (0, \Delta)$ are feedback gains, Δ_1 is a constant parameter representing the input saturation, to be chosen such that $\Delta_1 \leq \Delta - k_2$, and $\alpha \in (0, 1)$. The nonlinear term $\textsc{Tanh}(v_i)$ is applied to smoothly limit the magnitude of the control input, particularly for larger values of v_i, ensuring that the control law remains bounded even for significant deviations. The function $\textsc{Sat}_{\Delta_1}(x)$ denotes

the *saturation function*, which imposes a constraint on the classic protocol v_i to account for practical limitations in real-world systems, such as actuator limits. While this constraint helps prevent the closed-loop system from exceeding its operational bounds, it also inherently limits the system's performance by reducing the maximum achievable control effort. The saturation function is defined as:

$$\text{SAT}_\phi(Z) = \begin{cases} Z, & \text{if } |Z| \leq \phi, \\ \phi \cdot \text{SGN}(Z), & \text{if } |Z| > \phi. \end{cases}$$

where $\text{SGN}(\cdot)$ is the *sign function* given by:

$$\text{SGN}(Z) = \begin{cases} 1, & \text{if } Z > 0, \\ 0, & \text{if } Z = 0, \\ -1, & \text{if } Z < 0. \end{cases}$$

According to [42], the origin of the MAS in equation (5.5) is considered globally practically finite-time stable in terms of reaching consensus, namely its states reach consensus $\bar{x}$ within finite time if there exists a Lyapunov-Like function $\hat{V}(x)$ that satisfies positive definiteness and radial unboundedness properties 1 and 3, and the following adjusted condition on the Lie derivative:

4. The Lie derivative of $\hat{V}$ along the system's trajectories must satisfy the following inequality:

$$\dot{\hat{V}}(x) \leq -c \cdot \hat{V}^\alpha(x) + \delta, \quad c > 0, \quad \delta \in (0, +\infty), \quad \alpha \in (0, 1).$$

Moreover, if conditions 1, 3 and 4 hold, the convergence time T is upperbounded by:

$$T \leq \mathcal{T} = \left\lceil \frac{\max_{x_{i,0} \in x_0} \left\{ \hat{V}^{1-\alpha}(x_{i,0}) \right\}}{c \cdot \theta \cdot (1 - \alpha)} \right\rceil, \quad \theta \in (0, 1) \tag{5.6}$$

The new condition 4 highlights a relaxation of the classical Lyapunov stability criterion 2, offering a more flexible framework to ensure finite-time convergence in the considered MAS, since it allows $\dot{\hat{V}}(x)$ to be positive for certain values of x, as long as the inequality is satisfied overall.

5.2 Physics-Informed Neural Networks

Physics-informed neural networks (PINNs) represent a class of NNs that incorporate physical laws into the learning process, embedding differential equations as constraints within the network structure. Unlike traditional NNs, which learn purely from data, PINNs integrate prior knowledge of physical systems-such as conservation laws, governing equations, or boundary conditions-into the loss function. The latter approach was introduced by Raissi et al. [47] as a novel method to solve forward and inverse problems which are governed by Partial Differential Equations (PDEs), equations involving functions of multiple independent variables, such as space and time, and their partial derivatives. The idea consists in augmenting the standard

NN loss function with terms that penalize deviations from known physical laws. In this way, PINNs can learn both from the data and from the underlying physics, thereby providing a flexible and powerful tool for modeling complex systems.

PINNs can have applications in various fields. They have been successfully employed in fluid mechanics for challenging tasks like reconstructing three-dimensional wake flows and solving high-speed flow problems [6]. Additionally, PINNs have been applied also to enhance power system models, enabling the determination of dynamic states like rotor angles and frequency significantly faster than traditional methods [38]. These innovations are particularly valuable in scenarios requiring quick adjustments, such as renewable energy integration and grid stability management.

Regarding finite-time consensus for decentralized anomaly detection, the aim is to reach an agreement in finite time by all agents in a multi-agent system. This is done by seeking an appropriate Lyapunov-like function which satisfies conditions 1, 3 and 4 as defined in Section 5.1. Up to now, such functions are derived analytically, which can be challenging and infeasible for complex systems or when dealing with unknown dynamics. In this work the PINNs are innovatively applied to automatically learn a Lyapunov-like function directly from data for any given dynamical system, without requiring explicit analytical derivations. By embedding the necessary conditions for finite-time consensus into the NN's loss function, PINNs can efficiently learn a function that respects these conditions. This makes them particularly suitable for the decentralized anomaly detection framework proposed in this work, where each agent needs to ensure convergence to a consensus value under time constraints.

In the presented implementation, the PINN model is structured as a sequential NN with a configurable number of layers and activation functions. The input is composed by the independent variables of the considered PDE. An *hard constraint* function is applied at the output $\hat{y}$ to enforce conditions that the loss function alone may not guarantee.

5.2.1 Application of PINNs for Learning a Lyapunov Function

Firstly, to illustrate the effectiveness of PINNs, two training simulations are provided, each focusing on single-agent systems ($K = 1$). The following simulations differ in their choice of the dynamical system $\mathbf{f}(x, u(x))$ and the internal network configuration. In both cases, since time is the only variable governing these systems, they can be described by Ordinary Differential Equations (ODEs), particular PDEs involving only one independent variable.

The PINNs are then designed to have the independent variable time t as input, while the learned output $\hat{y}$ is composed by the traditional Lyapunov function $V(x)$ and the control input $u(x)$, necessarily required to satisfy the Lyapunov closed-loop stability constraints:

$$\hat{y} = \mathcal{F}^{\text{PINN}}(t; W) = [V(x), u(x)]$$

The PINNs are trained in order to minimize the loss function $\ell(D, W)$, defined in (5.7), which expresses the classical Lyapunov stability criterion. Given as inputs the learned parameters $\hat{y}$ (dependent as known on the model weights W) and a dataset D containing batches of state vectors $x \in \mathbb{R}^{\mu}$ randomly generated between some domain values dependent on the nature of the considered system, this loss function comprises three primary terms $L^p(D, W)$, $L^d(D, W)$, $L^u(D, W)$ corresponding to

the requirements for ensuring conditions 1, 2 and 3, respectively. The term $L^d(D, W)$ is particularly stringent, penalising values where the Lie derivative exceeds $-Q$, with $Q = 5$. The coefficients w^p, w^u, w^d are positive weights assigned to each term, in order to balance their relative importance in the overall loss.

$$\begin{cases} \ell(D, W) = w^p \cdot L^p(D, W) + w^u \cdot L^u(D, W) + w^d \cdot L^d(D, W), \\ L^p(D, W) = \text{ReLU}\Big(\|x\|_2 - V(x)\Big)^2, \\ L^u(D, W) = \text{ReLU}\Big(V(x) \cdot \|x\|_2\Big)^2, \\ L^d(D, W) = \text{ReLU}\Big(\dot{V}(x) + Q \cdot \|x\|_2\Big)^2 \end{cases} \tag{5.7}$$

In these expressions, the Rectified Linear Unit function, which is defined as $\text{ReLU}(Z) = \max\{0, Z\}$, returns the input value Z if it is positive and zero otherwise. It effectively applies a threshold at zero, capturing any positive deviations in the loss terms while ignoring negative ones. The constraint $V(0) = 0$ is enforced through the hard constraint function applied at the output layer: it takes the predicted output $\hat{y} = [V, u]$ and multiplies the input state x with the predicted Lyapunov function, thereby annihilating the value of $V(x)$ when the input is zero. The described PINN training framework is illustrated in Figure 5.1.

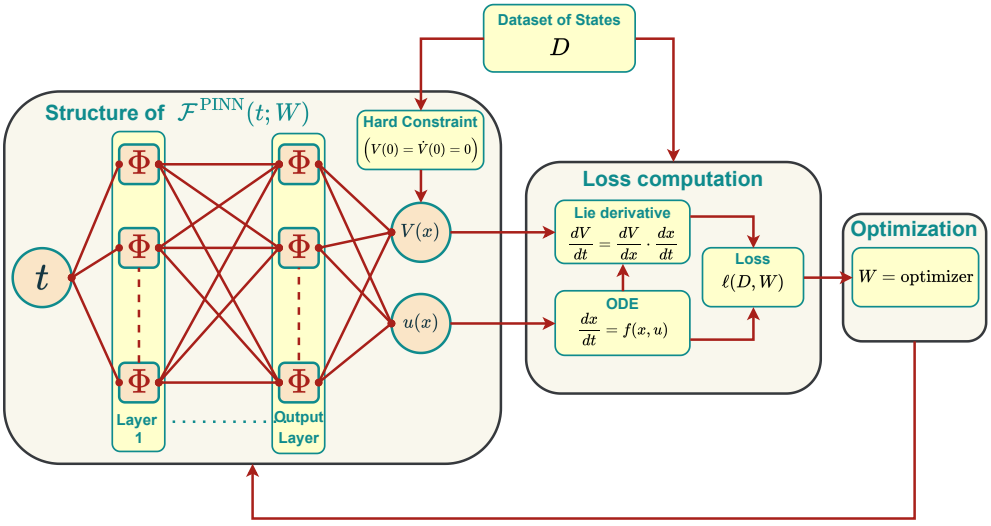


Figure 5.1 Physics-Informed Neural Networks training algorithm

Learning Lyapunov for Integrator System The first reported simulation considers a one-dimensional ($\mu = 1$) integrator system:

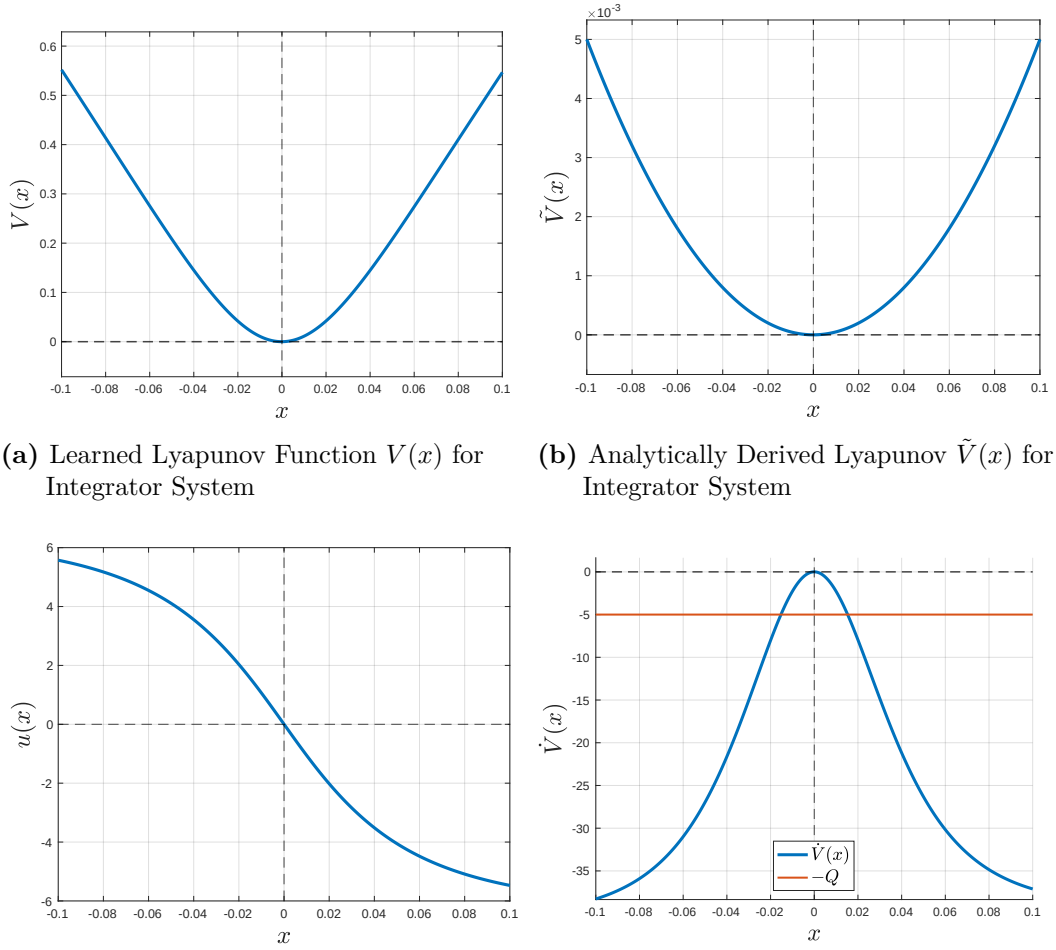
$$\mathbf{f}(x, u(x)) = u(x), \quad x \in \mathbb{R} \tag{5.8}$$

The objective of the PINN is to learn both the Lyapunov function and the control input that stabilize the system at the origin. The network is trained for 100 epochs

using a PINN architecture with 3 layers, each containing 100 nodes, and a TanH activation function. The loss function weights are set as $w^p = 4, w^u = 1, w^d = 1$ to balance the contributions of the positivity, unboundedness, and derivative constraint terms, respectively. The dataset D contains 10.000 randomly generated states with values ranging between $x \in [-6.0, 6.0]$.

To evaluate the effectiveness of the learned Lyapunov function, is presented a comparison with the analytically derived standard quadratic Lyapunov function (5.9) for the integrator system.

$$\tilde{V}(x) = \frac{1}{2} \cdot x^2 \tag{5.9}$$



(a) Learned Lyapunov Function $V(x)$ for Integrator System

(b) Analytically Derived Lyapunov $\tilde{V}(x)$ for Integrator System

(c) Learned Control Input $u(x)$ for Integrator System

(d) Lie Derivative of Learned Lyapunov $V(x)$ for Integrator System

Figure 5.2 Trained PINN Outputs and relative Lie derivative for the Integrator system, compared to the standard analytical Lyapunov Function

The results of this first simulation are displayed in Figure 5.2. As expected, Figures 5.2a and 5.2b show a close agreement between the learned and the analytically

derived Lyapunov functions, confirming that the PINN effectively approximates the analytic desired function.

Subsequently, the control input learned by the PINN and the Lie derivative of the learned Lyapunov function are examined to ensure they satisfy the necessary stability conditions. Figure 5.2c shows the control input learned by the PINN, from the analysis of which it is evinced that the control value at zero is indeed zero, indicating no action at the origin where equilibrium exists. While Figure 5.2d demonstrates that the Lie derivative remains negative and less than the displayed line $-Q$ for most of the state space, except within a small neighbourhood around the origin (since $\dot{V}(0)$ is enforced to be zero) thereby satisfying the conditions for robust stability.

Learning Lyapunov for Inverse Pendulum System The second simulation focuses on a more complex system: the inverse pendulum, which is modeled as a two-dimensional ($\mu = 2$) dynamical system, whose states are θ and ω, indicating respectively the angular position and velocity with respect to the downward vertical position $(\pi, 0)$. The state-space representation of the system is given by:

$$\mathbf{f}(x, u(x)) = \mathbf{f}\Big(\theta, \omega, u(\theta, \omega)\Big) = \begin{bmatrix} \dot{\theta} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \omega \\ \frac{-mgL\sin(\theta) - b\omega + u(\theta,\omega)}{J} \end{bmatrix} \tag{5.10}$$

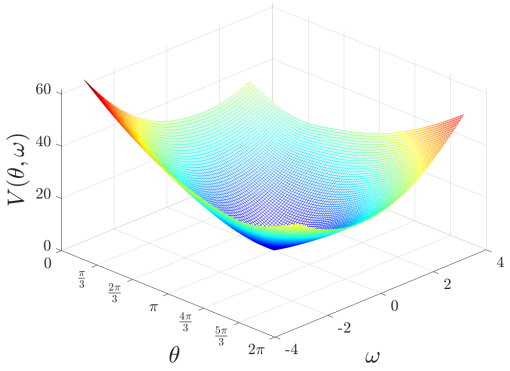
In this model, m represents the mass of the pendulum in kilograms, while g denotes the acceleration due to gravity. The length of the pendulum from the pivot point to its centre of mass is given by L in meters. The parameter b is the damping coefficient, which accounts for frictional forces acting on the pendulum and is measured in $\mathrm{kg}\cdot\mathrm{m/s}$. The moment of inertia of the pendulum about the pivot point is represented by J in $\mathrm{kg}\cdot\mathrm{m}^2$, while the parameter $u(\theta, \omega)$ is the control input applied to reach the desired state. In order to focus on stabilizing the pendulum in its upright position, a change of coordinates is performed. The angular position variable is redefined such that $\theta = \bar{\theta} + \pi$, where $\bar{\theta}$ represents the angular deviation from the upright equilibrium position. This transformation shifts the equilibrium point from $(\theta, \omega) = (\pi, 0)$ to the origin $(\bar{\theta}, \omega) = (0, 0)$. The state-space representation of the system in the new coordinates is given by:

$$\mathbf{f}\left(\bar{\theta}, \omega, u(\bar{\theta}, \omega)\right) = \begin{bmatrix} \dot{\bar{\theta}} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \omega \\ \frac{-mgL\sin(\bar{\theta}+\pi) - b\omega + u(\bar{\theta},\omega)}{J} \end{bmatrix}, \tag{5.11}$$

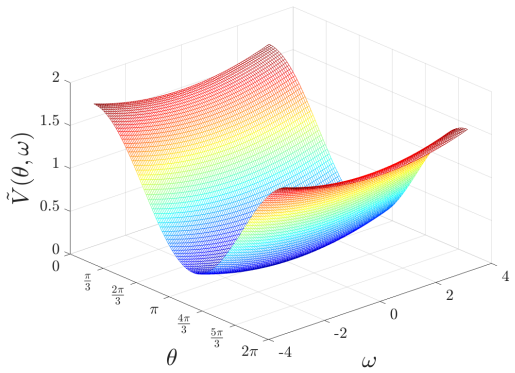
In this case, the PINN is designed to learn both the Lyapunov function $V(\bar{\theta}, \omega)$ and the control $u(\bar{\theta}, \omega)$ required to stabilize the pendulum in the upright position, corresponding to the shifted equilibrium point $(\bar{\theta}, \omega) = (0, 0)$.

A possible choice for an analytical Lyapunov function for the system 5.10 is:

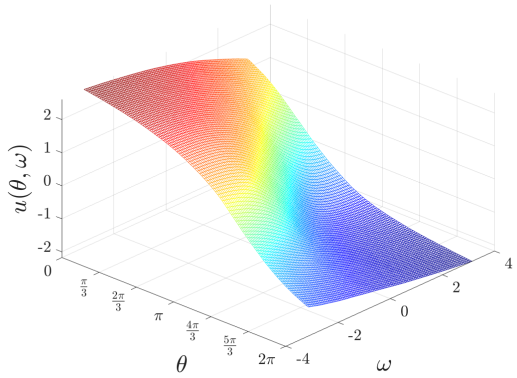
$$\tilde{V}(\theta, \omega) = \frac{1}{2}J\omega^2 + mgL(1 - \cos(\theta)) \tag{5.12}$$



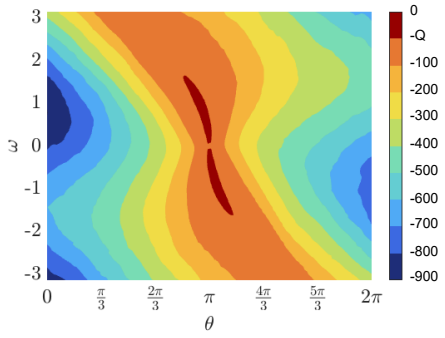
(a) Learned Lyapunov Function $V(\theta, \omega)$ for Inverse Pendulum



(b) Analytically Derived Lyapunov Function $\tilde{V}(\theta, \omega)$ for Inverse Pendulum



(c) Learned Control Input $u(\theta, \omega)$ for Inverse Pendulum



(d) Lie Derivative of the Learned Lyapunov $V(\theta, \omega)$ for Inverse Pendulum

Figure 5.3 Trained PINN Outputs and relative Lie derivative for the Inverse Pendulum system, compared to the standard analytical Lyapunov Function

Due to the complexity of the task, the network is now trained for 300 epochs with a configuration of 10 layers, each containing 500 nodes, and using a ReLU activation function. The hyperparameters of the model are set as follows: the acceleration due to gravity is $g = 9.81\,\mathrm{m/s^2}$, the length of the pendulum is $L = 0.5\,\mathrm{m}$, the mass of the ball at the end of the pendulum is $m = 0.15\,\mathrm{kg}$, the friction coefficient is $b = 0.1\,\mathrm{kg \cdot m^2/s}$, and the moment of inertia is $J = mL^2 = 0.0375\,\mathrm{kg \cdot m^2}$. Here, the best results were achieved by setting the loss function weights to $w^p = 4, w^u = 0.001, w^d = 1$. The dataset D contains now 10.000 randomly generated pairs of state values, with θ, ω uniformly distributed in the range $[-\pi, +\pi]$, to cover all possible positions and velocities of the pendulum. Figure 5.3 shows the results of this second simulation. To assess the accuracy of the learned Lyapunov function, Figures 5.3a and 5.3b present a comparison, in the original coordinates (θ, ω), with the analytically derived Lyapunov function for the inverse pendulum system. While $V(\theta, \omega)$ does not exactly match the analytical function $\tilde{V}(\theta, \omega)$, it closely approximates the desired behaviour by capturing the essential characteristics of the system's energy. Moreover the overall shape and stability properties are consistent, demonstrating the effectiveness of the PINN in approximating a Lyapunov function for the inverse pendulum system. Additionally, the control input generated by the PINN and the Lie derivative of

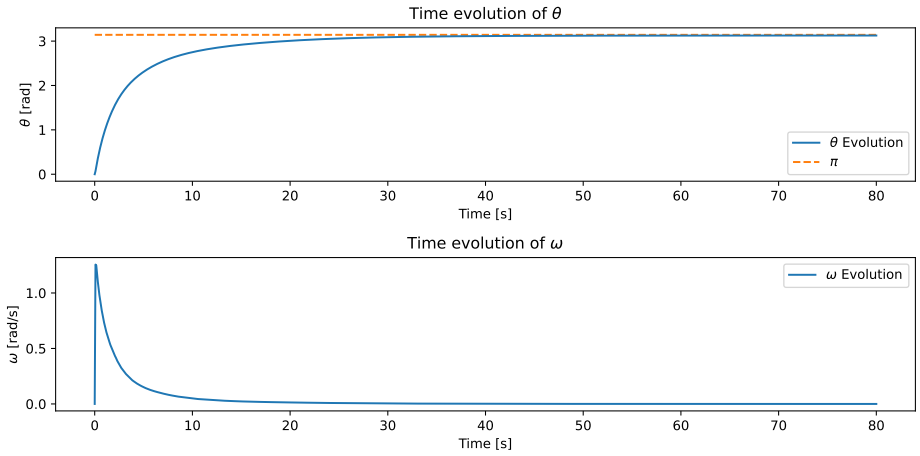


Figure 5.4 Time Evolution of Angular Position θ and Angular Velocity ω for the Learned Closed-Loop Control of the Inverse Pendulum

the corresponding Lyapunov function are examined to confirm compliance with the required stability conditions. Figure 5.3c presents the control input obtained from the PINN, while Figure 5.3d provides a top-down view of the Lie derivative $\dot{V}$, demonstrating that the closed-loop system achieves and maintains stability under the applied learned control strategy. In particular, the red region represents states where $-Q \leq \dot{V} < 0$, which do not satisfy the stringent robust derivative constraint but still guarantee stability. The rest of the plot, corresponding to states where $\dot{V} < -Q$, confirms robust stability, indicating that the control strategy effectively drives the system toward the desired stability conditions.

Finally, the time evolutions of the angular position θ and velocity ω are presented in Figure 5.4, solutions to the closed loop dynamical pendulum system starting from the resting condition, i.e., with initial conditions

$$\begin{bmatrix} \theta_0 \\ \omega_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

to further verify that the learned control input successfully stabilizes the pendulum. As shown in the figure, both the angular position and velocity converge to their respective equilibria over time, demonstrating that the closed-loop stability condition with the learned Lyapunov function is effectively ensured. The learned control strategy is capable of maintaining the pendulum in its upright position, thereby confirming the robustness and effectiveness of the PINN in approximating the desired stabilization behaviour.

5.2.2 Application of PINNs for Finite-Time Consensus

Having demonstrated the effectiveness of PINNs in learning traditional Lyapunov functions V for single-agent systems, their application is extended to learning a Lyapunov-like function $\hat{V}$ that satisfies finite-time convergence conditions for MAS with $K > 1$. The following results focus exclusively on the integrator MAS

(5.5), demonstrated to reach finite-time consensus according to [42] if satisfied the Lyapunov-Like conditions 5.1.2.

In this context, the objective shifts, from stabilizing the system at an equilibrium point, to reaching a consensus value $\bar{x}$, which represents the mean of the initial states of all agents. Consequently, the need for learning a control input for stability is eliminated, and the focus is placed solely on identifying a suitable Lyapunov-like function $\hat{V}$ and a corresponding α parameter that guarantee finite-time convergence to consensus.

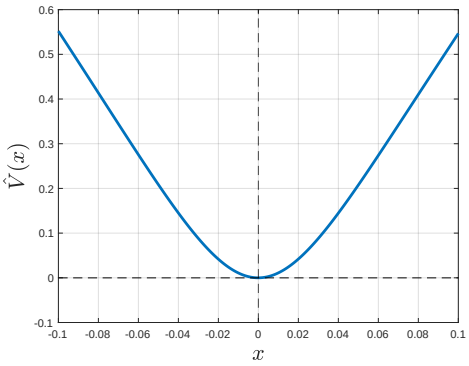


Figure 5.5 Learned Lyapunov-like Function for Finite-Time Consensus with $\alpha = \frac{21}{25}$

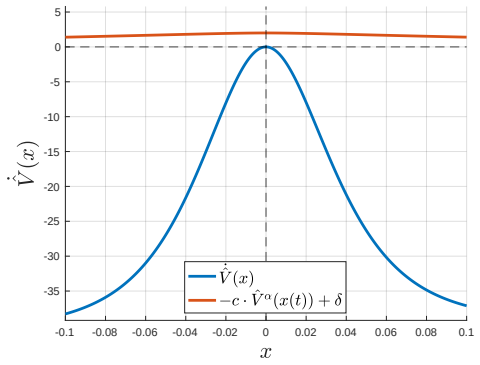


Figure 5.6 Lie Derivative of the Learned $\hat{V}$ for Finite-Time Consensus compared with the curve defined by Lyapunov-Like condition

To adapt PINNs for finite-time consensus, the modified loss function (5.13) is considered, which incorporates the finite-time convergence condition 4. Unlike the loss function defined in (5.7), this new formulation adjusts the derivative loss term to specifically penalise deviations from the desired finite-time convergence behaviour. Thus, the PINN training algorithm is the same described in Figure 5.1, except for the loss computation step. The parameter $\alpha \in (0, 1)$ for training is chosen as the ratio between two odd numbers, such that $\hat{V}^{\alpha}(x)$ can be computed even if $\hat{V}(x) < 0$.

$$\begin{cases} \hat{\ell}(D, W) = w^p \cdot L^p(D, W) + w^u \cdot L^u(D, W) + w^d \cdot \hat{L}^d(D, W), \\ \hat{L}^d(D, W) = \text{ReLU}\left(\dot{\hat{V}}(x) - (-c \cdot \hat{V}^{\alpha}(x) + \delta)\right)^2 \end{cases} \tag{5.13}$$

Figures 5.5 and 5.6 display respectively the Lyapunov-like function and its Lie derivative learned from a PINN trained for 100 epochs, under the loss function (5.13), configured with 10 layers composed each one by 500 neurons, and using a ReLU activation function. The chosen parameters for $\hat{L}^d(x)$ are $c = 1$, $\delta = 2$, $\alpha = \frac{21}{25}$, while the weighting terms are fixed as follows: $w^p = 4, w^u \equiv w^d = 0.1$. From their visual analysis, it can be stated that the output produced by PINN effectively satisfies the necessary conditions 1, 3 and 4 for Lyapunov-Like practical finite time convergence towards consensus.

To further demonstrate that the existence of the resulting $\hat{V}$ for a MAS of integrators effectively implies consensus within finite time, the consensus protocol (5.5) is performed over the communication topology $\mathcal{G}$ defined in Figure 5.7, composed by $K = 5$ agents.

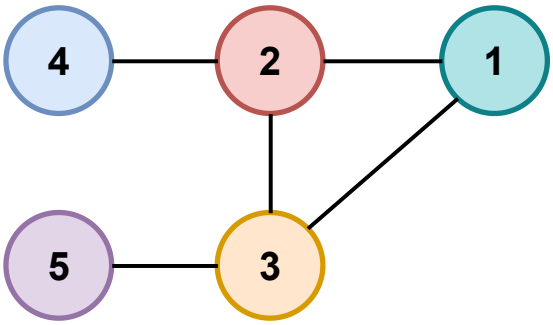


Figure 5.7 Considered Graph $\mathcal{G}$ for the Consensus Protocol

The collective dynamics of agents following the protocol (5.5) can be encapsulated in the control vector, coincident to the state dynamics of the multi-agent integrator system:

$$\dot{x} = u(x) = \text{SAT}_{\Delta_1}\left(k_1 \cdot (Lx)^{\alpha}\right) + k_2 \cdot \text{TANH}\left(Lx\right) \tag{5.14}$$

where L is the Laplacian matrix of the graph $\mathcal{G}$. The block scheme related to the closed loop MAS (5.14) is shown in Figure 5.8.

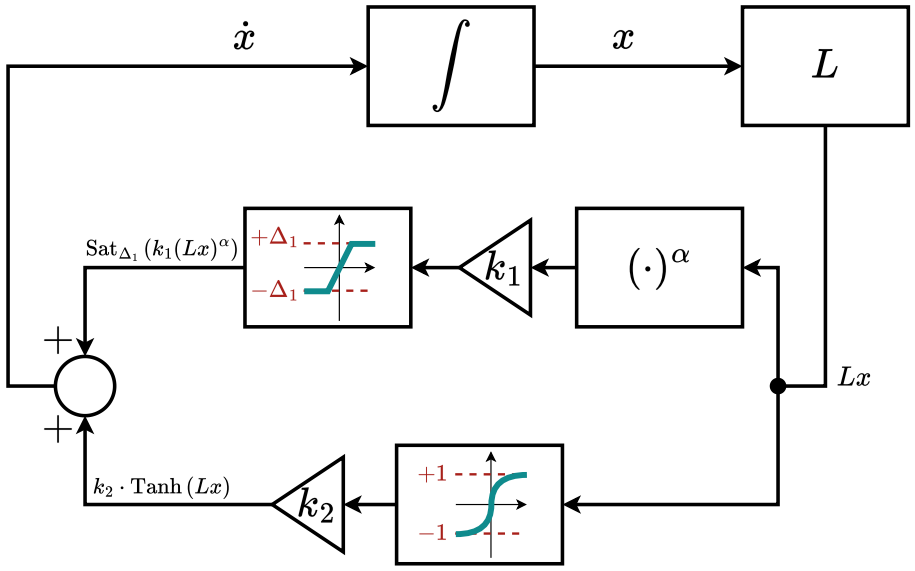


Figure 5.8 Block scheme of the considered MAS

The presented consensus protocol simulation is performed by fixing $\Delta = 5.4$ and the related parameters as $k_1 = 3.5$, $k_2 = 1.4$, $\Delta_1 = \Delta - k_2 = 4$. As mentioned before, it's crucial to use the same parameter $\alpha = \frac{21}{25}$ adopted for learning $\hat{V}(x)$, since condition 4 is satisfied for that specific value of α.

By defining a time step $t = 0.01s$, Figure 5.9, confirm that all agents in the system (5.5) successfully reach a common consensus value $\bar{x}$ within approximately $T \approx 50$ time steps $(0.5s)$, which is significantly shorter than the time window

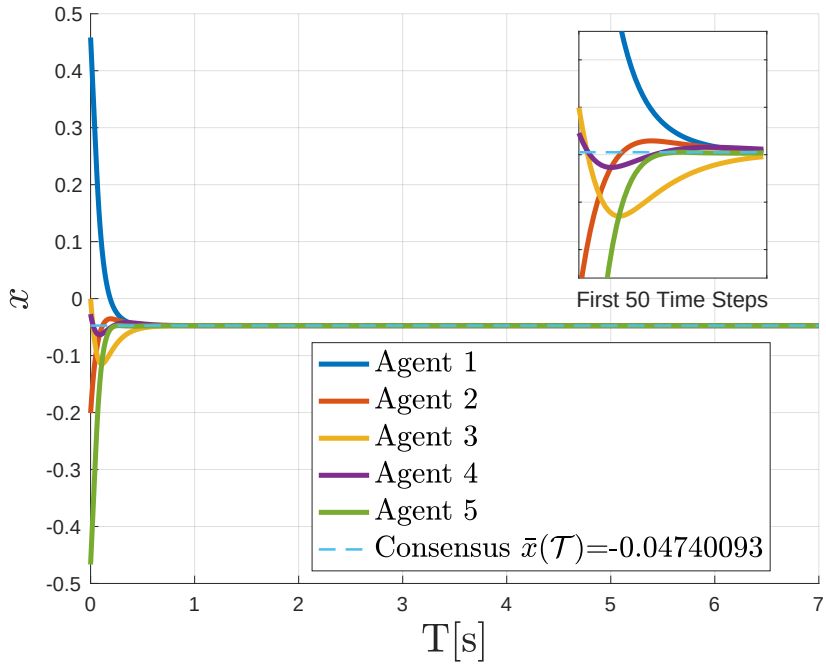


Figure 5.9 Time Evolution of Agents' States under Finite-time Consensus Protocol

chosen as the upper bound $\mathcal{T} = 7s$, established in (5.6) with the choice of $\theta = 0.9$. These findings confirm the effectiveness of the PINN-based approach in identifying a Lyapunov-like function that guarantees finite-time convergence, highlighting its potential for decentralized anomaly detection in MEC environments.

5.3 Proposed Consensus-based Anomaly Detection Algorithm

The finite-time consensus protocol, as discussed in previous sections, is now applied to decentralize the FL algorithm for log anomaly detection tasks explained in Chapter 4. This decentralized approach, termed DecAdaLightLog, refers to a MEC environment composed by K edge clients, whose topology is modeled by an undirected graph $\mathcal{G}$, as shown in Figure 5.10.

The preprocessing phase and the TCN model utilized for local training in this decentralized approach are identical to those used in AdaLightLog, as described in Sections 4.1 and 4.2, ensuring continuity in the methodology and leveraging established procedures to handle log data effectively. During each c-th communication round, every agent k, where $k = 1, \ldots, K$, trains its TCN model, parameterized by weight vector $W_k^c \in \mathbb{R}^M$, where M is the the number of weights in the model:

$$W_k^c = \begin{bmatrix} \chi_{k,1}^c \\ \chi_{k,2}^c \\ \vdots \\ \chi_{k,M}^c \end{bmatrix} \tag{5.15}$$

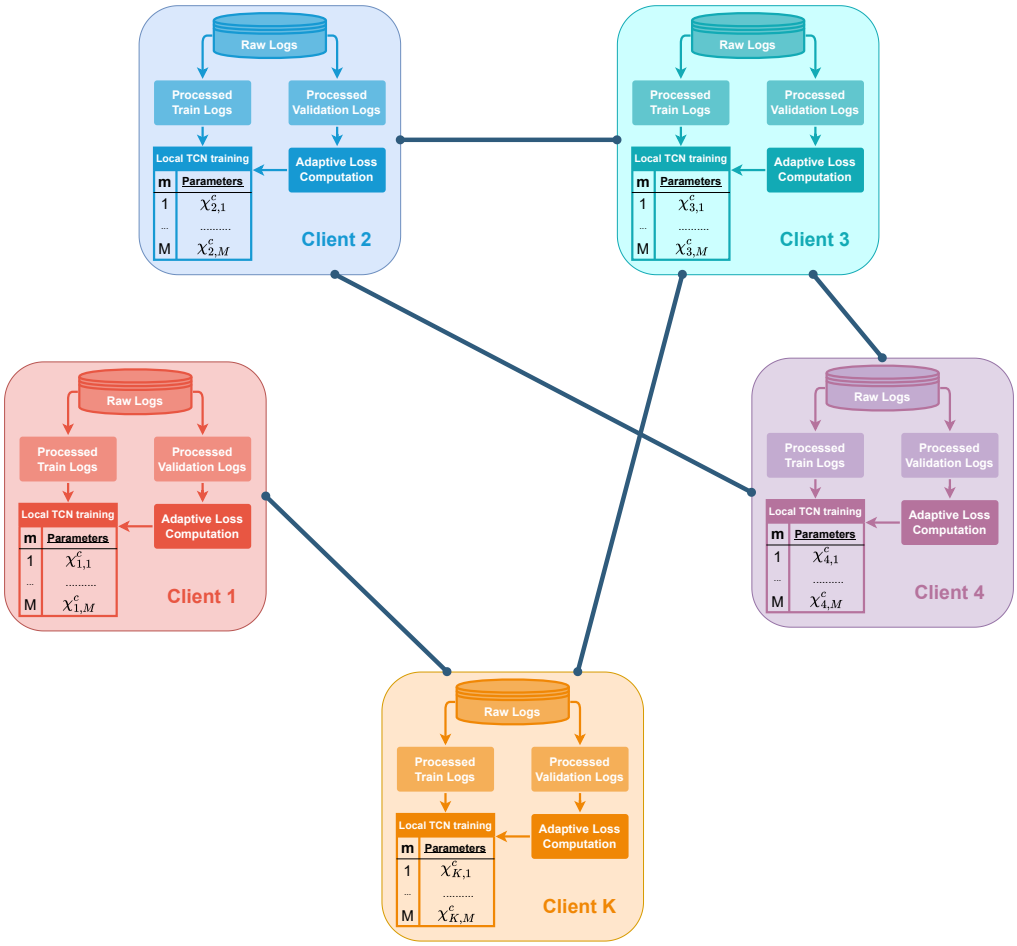


Figure 5.10 DecAdaLightLog framework

These parameters are updated on local processed data D_k, utilizing the same weighted categorical cross-entropy loss function (4.3) employed in AdaLightLog, replicating its benefits in a decentralized context.

The primary objective of generic FL algorithms is to synchronise the agent weights such that, in the subsequent communication round $c+1$, all K agents' NNs operate with the same parameters $\bar{\chi}_1^{c+1}, \ldots, \bar{\chi}_M^{c+1}$. Each m-th vector component $\bar{\chi}_m^{c+1}$ (where $m = 1, \ldots, M$) is equal to the average of weights $\chi_{1,m}^c, \ldots, \chi_{K,m}^c$ across all nodes, which is given by:

$$\bar{\chi}_m^{c+1} = \frac{1}{K} \sum_{k=1}^{K} \chi_{k,m}^c \tag{5.16}$$

In centralized settings, these weights are aggregated on a coordinating server that performs the averaging, as for AdaLightLog and FedAvg approaches. However, as proposed in this Section, the same outcome can be innovatively achieved in a decentralized manner by conducting M consensus protocols, one for each NN weight.

Firstly considering for each k-th agent a fixed generic **m**-th weight component

$\chi^c_{k,\mathbf{m}}$, a dynamical multi-integrator system is defined, comprising K agents interconnected according to the topology modeled by $\mathcal{G}$. Each weight component $\chi^c_{k,\mathbf{m}}$ represents the initial state of an integrator system (5.17) that evolves according to the consensus protocol (5.5). As demonstrated in Section 5.2.2, this protocol allows practical finite-time consensus (5.4) under the conditions described in Section 5.1.2. This guarantees that, for any time $\mathbf{t} \geq T$, $\xi^c_{1,\mathbf{m}}(\mathbf{t}), \ldots, \xi^c_{K,\mathbf{m}}(\mathbf{t})$, which represent the $\mathbf{m}$-th weights of the K agents, converge to values within a small neighbourhood around the required average $\bar{\chi}^{c+1}_{\mathbf{m}}$ (5.16) of the initial states $\chi^c_{k,\mathbf{m}}$.

$$\begin{cases} \dot{\xi}^c_{k,\mathbf{m}}(t) = u_{k,\mathbf{m}}(\xi^c_{k,\mathbf{m}}(t)), \quad \xi^c_{k,\mathbf{m}}(t) \in \mathbb{R}, \\ \xi^c_{k,\mathbf{m}}(0) = \chi^c_{k,\mathbf{m}} \end{cases} \tag{5.17}$$

Extending this concept, M multi-integrator systems are defined, one for each m-th weight component, characterized by its state vector $\xi^c_m(t) \in \mathbb{R}^K$ that evolves as follow:

$$\dot{\xi}^c_m(t) = \mathbf{f}_m(\xi^c_m(t)) = u_m(\xi^c_m(t)), \quad \xi^c_m(t) = \begin{bmatrix} \xi^c_{1,m}(t) \\ \xi^c_{2,m}(t) \\ \vdots \\ \xi^c_{K,m}(t) \end{bmatrix} \tag{5.18}$$

The protocols $u_1(\xi^c_1(t)), \ldots, u_M(\xi^c_M(t))$ ensure that all the K agents' trained parameters $W^c_k(\xi^c_{k,m}(t))$, now function of the states $\xi^c_{k,m}(t)$, converge towards consensus (5.19) corresponding to their element-wise average. Each agent k will adapt its resultant weight vector $W^c_k(\xi^c_{k,m}(\mathbf{t}))$ as the starting parameter configuration W^{c+1}_k for all agents in the subsequent communication round $c+1$, mirroring the classic FedAvg algorithm but executed in a distributed manner.

$$\begin{cases} \lim\limits_{t \to T} W^c_k(\xi^c_m(t)) = \lim\limits_{t \to T} \begin{bmatrix} \xi^c_{k,1}(t) \\ \xi^c_{k,2}(t) \\ \vdots \\ \xi^c_{k,M}(t) \end{bmatrix} = \begin{bmatrix} \bar{\chi}^{c+1}_1 \\ \bar{\chi}^{c+1}_2 \\ \vdots \\ \bar{\chi}^{c+1}_M \end{bmatrix} \quad \forall k = 1, \ldots, K \\ W^c_k(\xi^c_m(\mathbf{t})) = \begin{bmatrix} \bar{\chi}^{c+1}_1 \\ \bar{\chi}^{c+1}_2 \\ \vdots \\ \bar{\chi}^{c+1}_M \end{bmatrix} \quad \forall \mathbf{t} \geq T \end{cases} \tag{5.19}$$

To visually support this methodology, in Figure 5.11 is presented a toy example illustrating how weight synchronisation converges over time using the described finite-time consensus approach. This example features $K = 5$ agents, each with only $M = 4$ trained parameters, operating within the network topology shown in Figure 5.7 and the same hyper-parameters configuration of the example presented in Section 5.2.2. The considered simulation window is composed by $\mathcal{T} = 800$ time steps (8 seconds), chosen as the maximum upper bound time 5.6 among the 4 consensus processes. The figure clearly demonstrates that all convergences to consensus values are achieved in short time, dependent on the initial state configurations. The sparser these initial states are, the longer it takes to achieve consensus, as exemplified by the MAS with state ξ^c_2 that requires 80 time steps (0.8s) to convergence. This

simulation confirms that weight averaging can be effectively achieved within a totally decentralized context through consensus theory. Moreover, since the proposed control strategy guarantees that finite-time consensus, the reliability of weight averaging is always ensured. For better visualization, due to the fast convergence of weights, only 200 time steps are considered. The complete process of DecAdaLightLog FL training is reported in Algorithm 3.

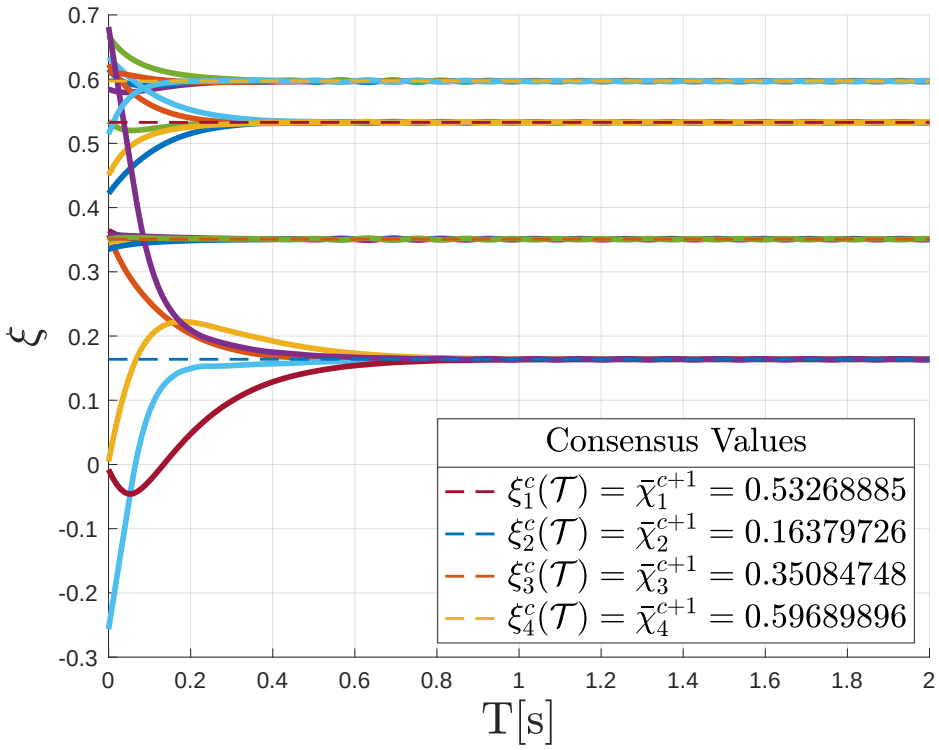


Figure 5.11 Execution of $M = 4$ consensus processes

Algorithm 3 DecAdaLightLog: Consensus-based FL Anomaly Detection Strategy

DecAdaLightLog:
Initialize $w_{0,k}^1$ and $w_{1,k}^1$
for each communication round $c = 1, \ldots, E$ **do**
  for each client $k = 1, \ldots, K$ **do**
    $W_k^c \leftarrow$ LocalTraining
  end for
  for each weight index $m = 1, \ldots, M$ **do**
    Setup communication graph $\mathcal{G}_m$ reflecting the MEC system's structure
    for each client $k = 1, \ldots, K$ **do**
      Define system $\dot{\xi}_{k,m}^c(t)$ as in (5.17), where $\xi_{k,m}^c(0) = \chi_{k,m}^c \in W_k^c$
      Assign $\dot{\xi}_{k,m}^c(t)$ to the k-th agent of $\mathcal{G}_m$
    end for
    Define MAS $\dot{\xi}_m^c(t)$ as in (5.18) related to $\mathcal{G}_m$
    repeat
      Update states $\xi_m^c(t)$ using consensus protocol (5.14)
    until $t = \mathbf{t} \geq T$
  end for
  for each client $k = 1, \ldots, K$ **do**
    $W_k^{c+1} \leftarrow \{\xi_{k,1}^c(\mathbf{t}), \ldots, \xi_{k,M}^c(\mathbf{t})\}$
    Compute the metric m_k^{c+1} by evaluating W_k^{c+1} on $k-th$ validation dataset
    Compute adaptive loss parameters $w_{0,k}^{c+1}$ and $w_{1,k}^{c+1}$ as in (4.4) and (4.5)
  end for
end for

Chapter 6

Simulation Results

Numerical simulations for both AdaLightLog and DecAdaLightLog were performed using the HDFS log dataset [54], which contains 11,175,629 entries, with approximately 2.58% of them marked as anomalies. The parsing method chosen for this analysis is Spell [37], known for achieving 100% parsing accuracy on the HDFS dataset. To maintain optimal parsing accuracy, the number of local servers is set to $K = 5$.

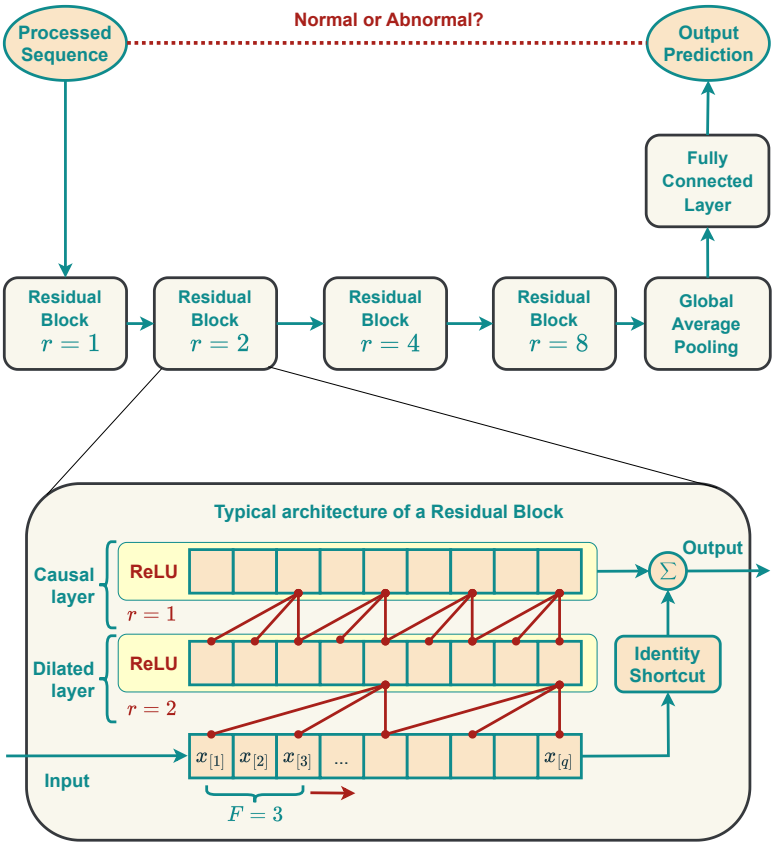


Figure 6.1 Illustration of the considered TCN

The dataset is then proportionally divided into K parts, each containing about $2,235,050 \pm 50$ log entries, ensuring a sufficient amount of data for effective parsing. Each dataset comprises approximately 21 ± 2 distinct log keys. All logs for each client are grouped into sequences based on their block ID. Sessions are labeled as abnormal (i.e., 1) if at least one log key within the session is identified as abnormal; otherwise, they are labeled as normal (i.e., 0). Clients have about 2.45% abnormal sessions, corresponding to roughly $180,000 \pm 30,000$ normal sessions and $4,500 \pm 500$ abnormal sessions, closely matching the percentage found in the original dataset.

Log embedding is then performed with WORD2VEC model to produce vectors of dimension $\mathcal{M} = 300$, as proposed by Wang et al. [59]. By eliminating $d = 7$ dominant eigenvectors, the PPA method reduces the embedding dimensionality to $n = N_5 = 19$, which corresponds to the smallest number of log keys among all local server datasets (specifically, in this case, the fifth local server); indeed, the number of components used for performing PCA cannot exceed the number of log keys N_i for a given k-th client.

Training is conducted over $E = 100$ global epochs. During each communication round, each local model undergoes training for $\tau = 3$ local epochs using the ADAM optimizer with a learning rate $\eta = 0.005$ and the adaptive loss function defined in (4.3). The adopted local TCNs, as illustrated in Figure 6.1, consist of four residual blocks with dilation rates set to $r = 1, 2, 4, 8$. Each block includes two convolutional layers: a dilated convolutional layer with its corresponding dilation rate and a causal convolutional layer. To both layers, after the convolution operation with fixed kernel size $F = 3$, the nonlinear RELU function is applied, while no pooling is applied. The output of the last residual block is reshaped by an Adaptive Average Pooling layer and a fully connected linear layer that outputs the binary classification of the log sequences. With this configuration, the TCNs are composed by $M = 5624$ parameters. As all clients at each communication round start training from the same set of weights and follow identical training procedures, the performances across clients are highly consistent. Consequently, the plots presented are based on the results from a single client (more precisely the fifth one), as they effectively represent the overall performance observed across all clients.

To assess the effectiveness of the considered fifth model, it is crucial to evaluate the predicted classes $\hat{Y}^5$ of a new validation dataset D^5_{test}, composed by input log sequences S^5 and their relative labels Y^5 unseen by the NN during the training process.

$$\hat{Y}^5 = \mathcal{F}_5^{\text{TCN}}(D^5_{\text{test}}; W_5^c) = \begin{bmatrix} \hat{y}_1^5 \\ \vdots \\ \hat{y}^5_{|D^5_{\text{test}}|} \end{bmatrix}, \quad D^5_{\text{test}} = \left(S^5, Y^5\right) = \left(\begin{bmatrix} S_1^5 \\ \vdots \\ S^5_{|D^5_{\text{test}}|} \end{bmatrix}, \begin{bmatrix} y_1^5 \\ \vdots \\ y^5_{|D^5_{\text{test}}|} \end{bmatrix} \right)$$

This comparison involves identifying instances where the model's predictions align with the true labels and where they diverge. The evaluation metrics used to quantify this performance are based on the following fundamental concepts:

- **True Positives (TP)**: Number of instances for which both the predicted and actual labels indicate an anomalous event.

$$TP = |\{j : \hat{y}_j^5 = 1 \wedge y_j^5 = 1\}|$$

- **True Negatives (TN)**: Number of instances for which both the predicted and actual labels agree on the absence of an anomaly.

$$TN = |\{j : \hat{y}_j^5 = 0 \wedge y_j^5 = 0\}|$$

- **False Positives (FP)**: Number of instances for which the model incorrectly predicts an anomaly, providing a false alarm.

$$FP = |\{j : \hat{y}_j^5 = 1 \wedge y_j^5 = 0\}|$$

- **False Negatives (FN)**: Number of instances for which the model fails to detect an actual anomaly, leading to missed detections.

$$FN = |\{j : \hat{y}_j^5 = 0 \wedge y_j^5 = 1\}|$$

These foundational counts are employed to compute the following considered evaluation metrics, that offer vital insights into the model's performance, highlighting various aspects essential for a robust anomaly detection system:

- **Accuracy**: Provides a broad measure of the model's overall prediction correctness, calculated as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision**: Measures the model's ability to avoid false positives and is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall**: Measures the model's ability to avoid false negatives, crucial for ensuring no anomalies are missed.

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1-Score**: Provides an harmonic mean of precision and recall, especially useful in scenarios with uneven class distribution.

$$\text{F1-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Among these metrics, Recall is particularly relevant due to its direct impact on the model's capacity to identify all actual anomalies. A high recall ensures that the model detects nearly all anomalies, which is critical for maintaining system integrity and preventing potential operational failures. Conversely, a low recall indicates a higher rate false negatives, which could lead to critical system issues going unnoticed. This can severely compromise the system's operational safety and effectiveness, as it allows real threats to pass undetected. While Precision is also important, its role is notably different. Precision measures the accuracy of the positive predictions made

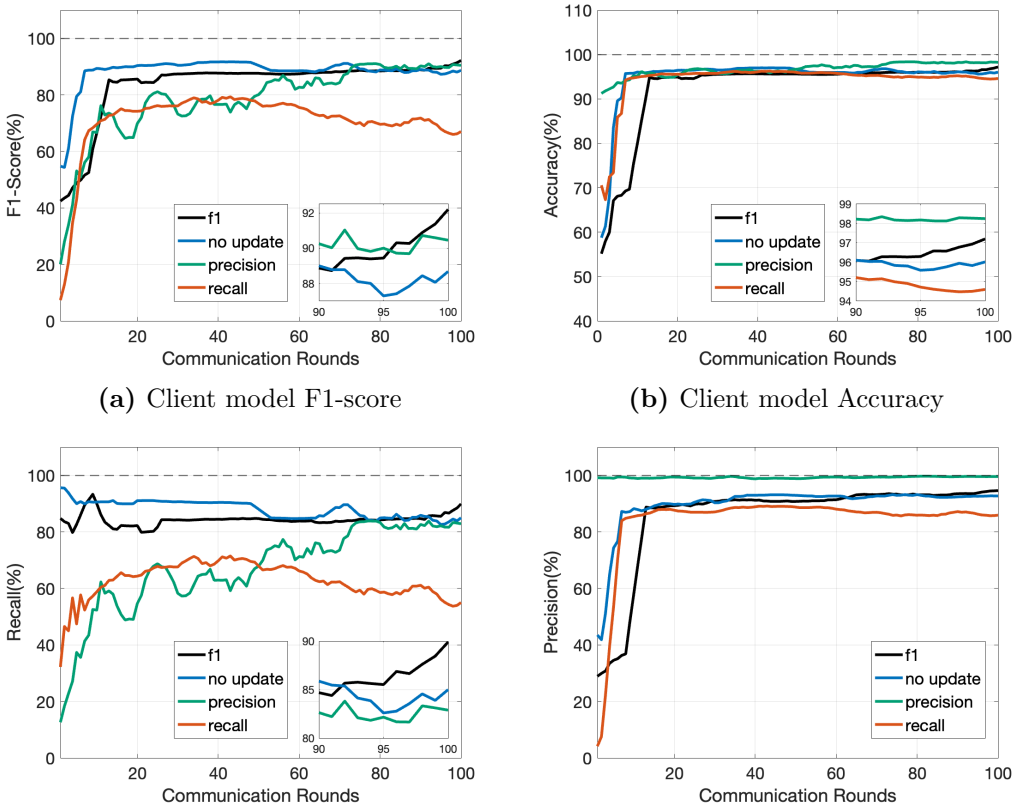
by the model. A lower precision leads to a higher number of false positives, reducing operational efficiency as it necessitates additional verification processes and results in unnecessary alerts. However, despite these inefficiencies, the model still captures critical anomalies, which can be crucial for initial screenings in systems where safety is the priority.

The following results will compare different model outputs with respect to the same validation dataset

6.1 AdaLightLog Results

Figure 6.2 shows the results of such comparison between the performance of the algorithm under a different adaptive loss updating rules. In order to highlight the effectiveness of the introduced adaptive loss, one model does not updates dynamically the loss weights, while the others vary the performance metric considered for the computation of the value m_5^{c+1}, necessary for the update of loss parameters $w_{0,5}^{c+1}$ and $w_{1,5}^{c+1}$. The considered performance metrics for updating m_5^{c+1}, as previously defined, include F1-score (a), Accuracy (b), Recall (c), and Precision (d) for the dedicated test set. For all the simulations, the average of parameters is weighted according to the performances s_k^c of each client with respect to the common central server validation dataset D_g, using the same metric Accuracy.

As it is possible to notice from Figure 6.2b, Accuracy achieves very high values across all models, with Precision metric-based model slightly outperforming others. However, as the dataset is highly imbalanced, Accuracy alone isn't a reliable metric for evaluating performance. The model using Recall demonstrates the poorest performance; its Recall (and F1-score) initially increases but later declines, potentially due to *overfitting*, a phenomenon that occurs when the model learns too closely from the training data, including its noise and outliers, which compromises its ability to generalize to new, unseen data. This results in high performance on training data but poor performance on test data or in real-world applications [49]. In contrast, the model based on F1-score exhibits the most stable convergence and best performance levels across all metrics. The other two models also perform well, albeit slightly behind the F1-score-based model. This first analysis suggests that adaptive loss functions AdaLightLog framework slightly improve overall performance particularly when not based on the Recall metric.



(a) Client model F1-score

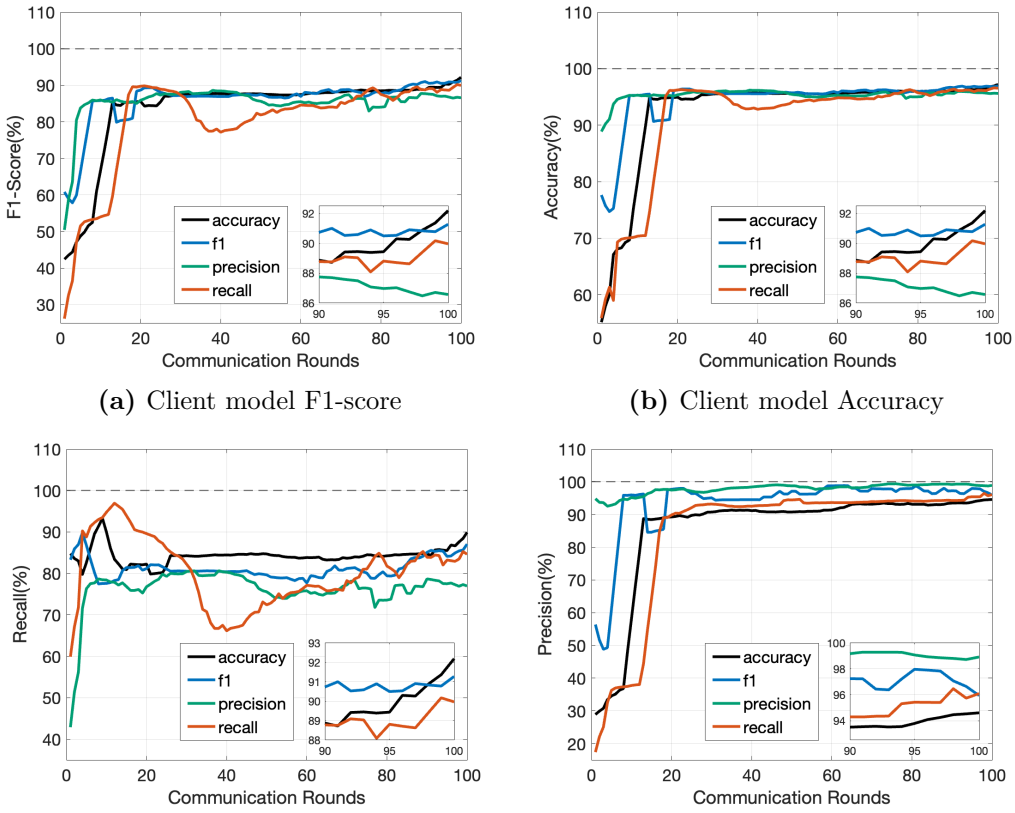
(b) Client model Accuracy

(c) Client model Recall

(d) Client model Precision

Figure 6.2 Comparison of four different trained AdaLightLog models, where three of them have performance metrics m_5^{c+1} for adaptive loss function update, while the fourth one has a static loss function. All the models use the same performance metric score s_5^c = Accuracy for weighting averaging

As the most promising performance metric for the update value m_5^{c+1} seems to be the F1-score, a second comparative analysis is performed by fixing it, now related to the local client evaluation score s_5^c varying among the Accuracy, F1-score, Precision and Recall. Figure 6.3 shows the results of such comparison, demonstrating outstanding performances for all the considered cases. The accuracy criteria is the most stable one over time and outperforms the others over all the comparative metrics (a)-(b)-(c), except over the Precision one (d), where it is surpassed by the Precision and Recall-based models of 4% and 1%, respectively.



(a) Client model F1-score **(b)** Client model Accuracy

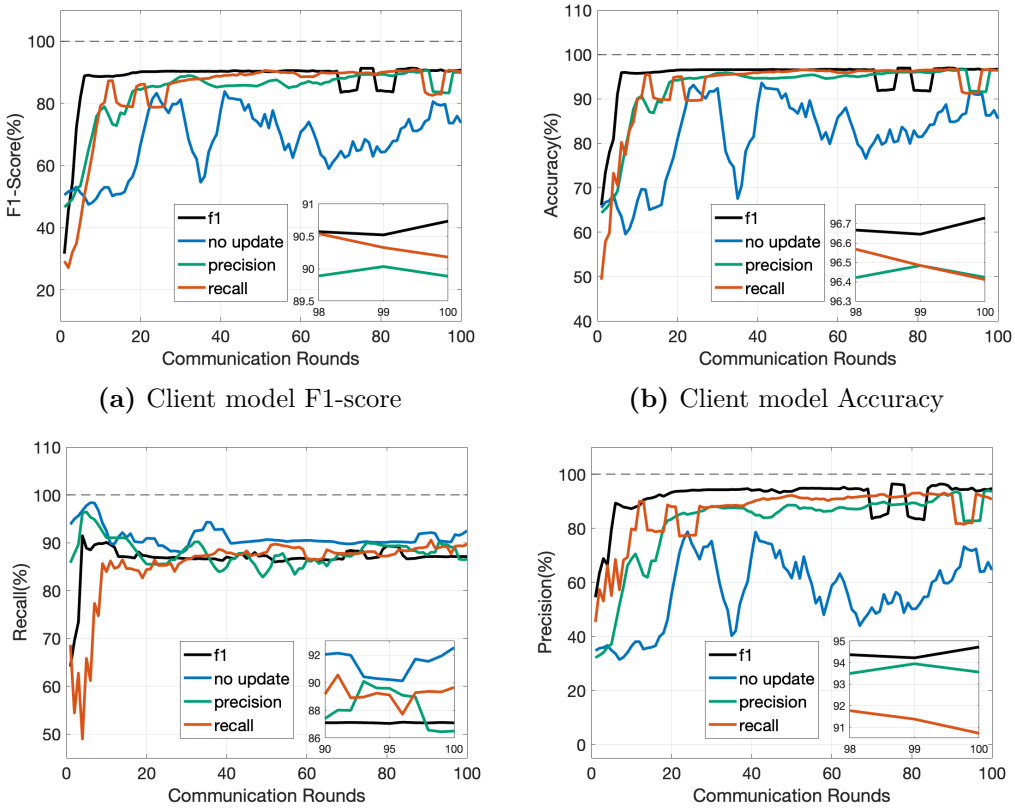
(c) Client model Recall **(d)** Client model Precision

Figure 6.3 Comparison over four AdaLightLog models, varying by their performance scores s_5^c of for weighted model averaging, keeping fixed performance metric for adaptive loss update m_5^{c+1} = F1-Score

6.2 DecAdaLightLog Results

Referring now to the proposed distributed approach DecAdaLightLog, since it is not involved a weighting average procedure, only four simulations are performed and compared. Different models are considered, varying for the adaptive loss updating rules. As for the first set of simulations in AdaLightLog, one model does not updates dynamically the loss weights, while the others vary the performance metric considered for the computation of the value m_5^{c+1}. In this case, as demonstrated in Figure 6.4, the absence of a dynamic update rule of the loss weights leads to several performance issues. In fact the non-adaptive model shows high instability during training epochs and poorest overall performance for all the considered metrics, except for the Recall that surprisingly has the highest value.

In contrast, the adaptive models exhibit very similar results, maintaining consistent accuracy, precision, recall, and F1-score throughout the training process, with minor oscillations in the last epochs due to the absence of regularization terms such as of learning rate η decay, dropout techniques or batch normalization.



(a) Client model F1-score

(b) Client model Accuracy

(c) Client model Recall

(d) Client model Precision

Figure 6.4 Comparison of four trained DecAdaLightLog models; three of them have different performance metrics m_5^{c+1} for adaptive loss function update, while the fourth model does not update its loss

As for the centralized approach, the most effective performance metric for the update value m_5^{c+1} seems to be the F1-score, that achieves slightly superior results in F1-Score, Precision and Accuracy, highlighting its balanced approach to managing false positives and false negatives.

Overall, the results indicate that incorporating an adaptive loss function, especially one based on F1-score, significantly enhances model performance in a decentralized learning environment. The adaptive models outperform the static loss model in nearly all metrics, confirming the benefits of a dynamic approach to loss weighting in Federated Learning frameworks like DecAdaLightLog.

6.3 Comparative Results

Finally, having found that for both approaches the m_5^c = F1-score is the most promising metric for the adaptive loss computation, and most in particular for AdaLightLog s_5^c = accuracy is the best for the weighted averaging, the resulting proposed algorithms AdaLightLog and DecAdaLightLog with their optimal configuration are compared with the standard FedAvg approach [36].

The results of such comparison are depicted in Fig. 6.5, with both AdaLightLog

and DecAdaLightLog outperforming FedAvg over all the considered performance metrics. Particularly focusing on the key metrics Recall and F1-score, while FedAvg achieves reasonable performance, these values lag behind the proposed methods, indicating that it struggles more with false negatives, which is critical in anomaly detection. The superior performance of AdaLightLog and DecAdaLightLog demonstrates the effectiveness of the introduced performance-based weighted averaging and dynamic loss function in identifying anomalies more accurately, enhancing their reliability for the log anomaly detection task.

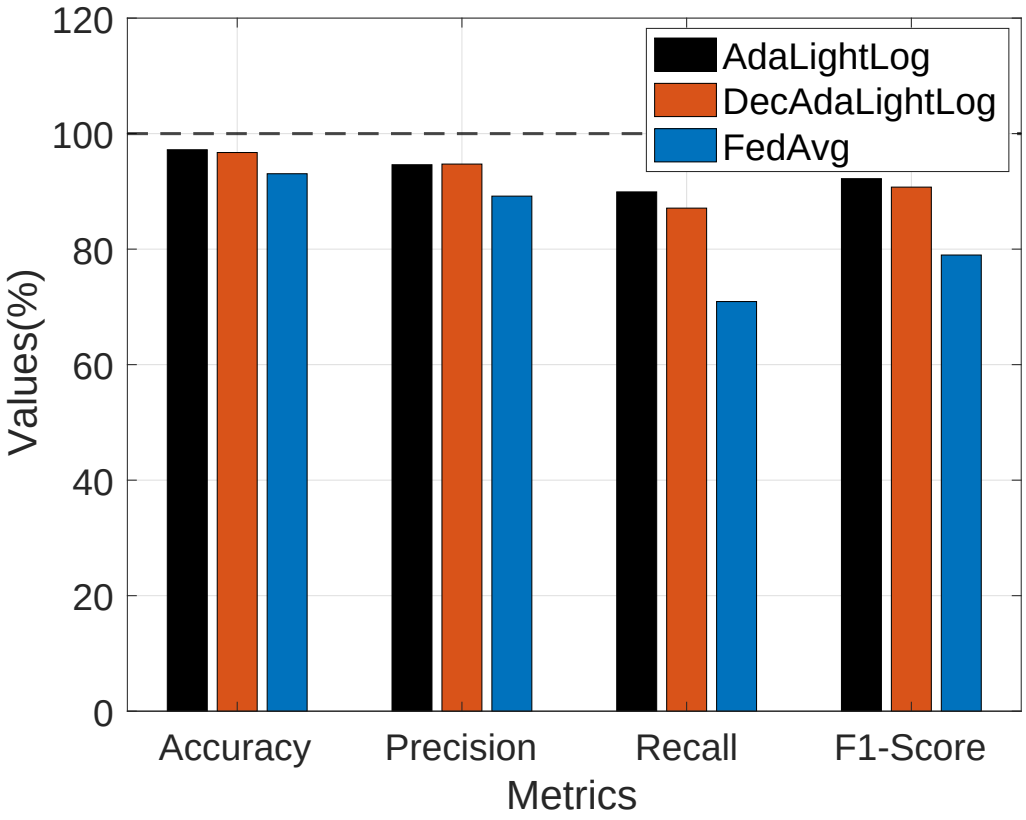
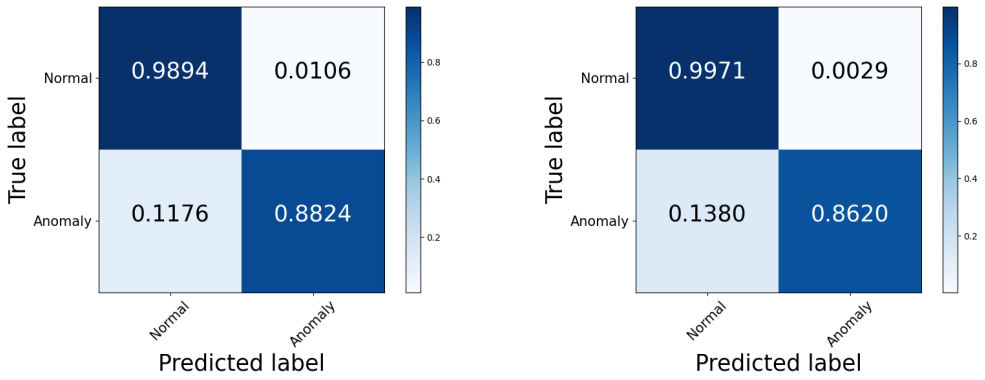


Figure 6.5 Performance comparison: AdaLightLog, DecAdaLightLog and FedAvg

The high performances of AdaLightLog and DecAdaLightLog are remarkably similar across all metrics, with AdaLightLog showing a very slight advantage. This minor difference can be attributed not only to the centralized weight averaging performed by AdaLightLog, which typically ensures a smoother aggregation process, but also to the fact that the centralized implementation employs weighted averaging of parameters based on the performances (accuracies in this case) of each client. In contrast, DecAdaLightLog cannot utilize weighted averaging, as it would require a common validation dataset which conflicts with the decentralization principles. Despite this, DecAdaLightLog achieves nearly identical results, proving practicality and efficiency of consensus-based methods for decentralized FL, which are more suitable for real-world distributed applications such as MEC environments where centralized model aggregation can be inefficient.



(a) Confusion matrix of AdaLightLog model

(b) Confusion matrix of DecAdaLightLog model

Figure 6.6 Comparison of the performances between AdaLightLog and DecAdaLightLog models; each row of the confusion matrix is normalized

This results can be demonstrated analyzing also the *Confusion Matrices* of the three models. A confusion matrix for this specific binary classification task is a 2×2 table that shows the number of correct and incorrect predictions made by the model, broken down by each class (Normal, Abnormal). The four components of such a matrix are the the true positives and true negatives (on the main diagonal), false positives and false negatives (outside the diagonal). These values are computed, for each trained NN, by considering the true labels Y^5 of the validation dataset of 5-th client, common for all the three models to compare and composed by $|D^5_{\text{test}}| = 5879$ input-output pairs (of them, 4799 are classified as normal, 1080 are anomalous), and their different predictions $\hat{Y}^5$.

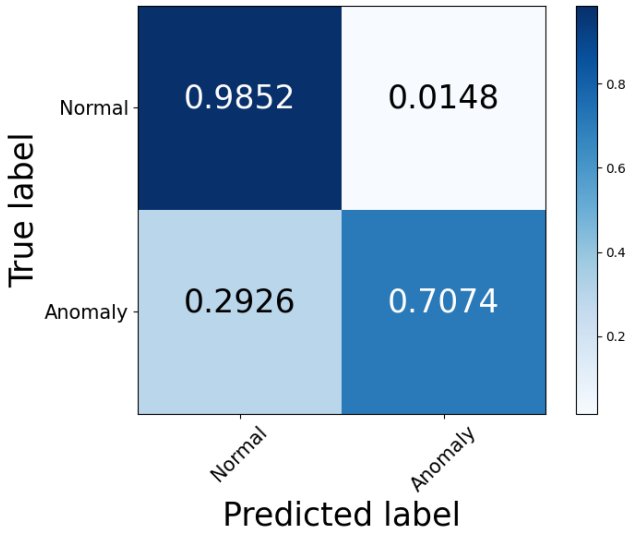


Figure 6.7 Confusion Matrix of the benchmark FedAvg algorithm

The confusion matrices for the proposed algorithms are shown in Figure 6.6. AdaLightLog reveals 4748 true negatives, 51 false positives, 953 true positives

and 127 false negatives. It means that 98.94% of normal sequences are correctly classified and 88.24% of anomalous sequences are detected. These percentage values are reported in Figure 6.6a. The same reasoning can be done for DecAdaLightLog, that results in 4785 true negatives, 14 false positives, 931 true positives and 149 false negatives. This distributed algorithm seems to perform better for correctly classifying normal sequences (correctly classified 99.71% of normal samples), while, as previously anticipated, slightly struggles to detect anomalies of about 2% with respect to the centralized approach (86.2% of anomalies correctly detected). Such percentages are reported in Figure 6.6b.

Figure 6.7 reports the confusion matrix of the traditional centralized FedAvg algorithm, characterized by 4728 true negatives, 71 false positives, 764 true positives and 316 false negatives. This standard approach still well classifies most of the normal sequences (98.52%) but, as demonstrated by Figure 6.5, almost 30% of the anomalous anomalous log sequences are undetected, in contrast with 11.76% and 13.8% of the presented methods.

Chapter 7

Conclusions and Future Work

This Thesis presented two innovative Federated Learning approaches for automatic log anomaly detection in Mobile Edge Computing environments, designed to enhance privacy by ensuring that clients do not need to share their raw data. The proposed frameworks, named AdaLightLog and DecAdaLightLog, address key challenges in federated anomaly detection while introducing several important innovations.

A first contribution is the introduction of an adaptive loss function for both AdaLightLog and DecAdaLightLog. This adaptive mechanism dynamically adjusts the importance of misclassified samples, improving the model's ability to learn effectively from more complex or challenging data instances. As a result, it prioritizes errors where the model struggles most, leading to an enhanced ability to detect anomalies across diverse data.

In the AdaLightLog framework, another notable innovation is the use of dynamic weighted parameter averaging. Here, each client's contribution to the global model update is based on its own performance, computed through validation metrics. This strategy not only improves the overall model performance but also increases robustness against potential attacks, as it reduces the influence of clients characterized by poor or potentially malicious behavior. This method of weighted averaging enables more reliable client contributions to guide the global learning process, significantly enhancing the system's adaptability and security.

The second framework, DecAdaLightLog, extends this concept by eliminating the need for a central coordinating server in order to address the single point of failure issue present in traditional FL systems, thus improving privacy and security. This is achieved by decentralizing the learning process through a finite-time consensus protocol. During the learning process, the MEC environment is considered as a multi-agent system, where agents exchange their model parameters, treated as the state variables of an integrator system, with neighboring clients to reach a consensus on the final model. This consensus process ensures that all clients agree on a common set of model weights after a finite number of communication rounds, facilitating distributed learning without the need for centralized control.

The innovative use of Physics-Informed Neural Networks within this context allows for the automatic learning of Lyapunov functions, guaranteeing finite-time consensus without the need for complex analytical derivations.

Comparative analyses were performed with the traditional FedAvg method, where both AdaLightLog and DecAdaLightLog showed superior performance in terms of accuracy, precision, recall, and F1-score. These evaluations included exploring different validation metrics for the adaptive loss function and the dynamic weight calculations. The results demonstrated that (i) the adaptive strategies introduced in this work lead to improved model robustness and effectiveness, with the proposed methods consistently outperforming FedAvg across the board, (ii) the proposed decentralized approach achieves nearly identical performance to the centralized AdaLightLog, thereby highlighting DecAdaLightLog as a viable solution for scenarios where full decentralization is preferred, such as environments where privacy, security, and the elimination of a central point of failure are critical.

These contributions represent a significant advancement in the field, offering a more secure, adaptive, and effective approach to federated anomaly detection.

While this work did not reveal significant performance differences between AdaLightLog and DecAdaLightLog, the weighted averaging strategy remains a crucial element in enhancing model robustness, particularly in scenarios involving potentially malicious clients. A promising direction for future work would be to implement a weighted averaging mechanism for DecAdaLightLog. This would involve developing a method to perform weighted averaging despite the decentralized nature of the problem, which inherently lacks a central coordinating entity. Achieving this could mitigate the risk of a malicious attack compromising a single client and subsequently degrading the overall performance of the system, thereby enhancing the security and reliability of the decentralized federated learning process.

Bibliography

[1] Rodolfo Stoffel Antunes, Cristiano André da Costa, Arne Küderle, Imrana Abdullahi Yari, and Björn Eskofier. Federated learning for healthcare: Systematic review and architecture proposal. *ACM Transactions on Intelligent Systems and Technology*, 13(4):1–23, May 2022.

[2] Daniel Barbará and Sushil Jajodia, editors. *Applications of Data Mining in Computer Security*. Springer US, 2002.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. In *Transactions of the Association for Computational Linguistics*, volume 5, pages 135–146, 2017.

[4] Aleksandar Botev, Guy Lever, and David Barber. Nesterov's accelerated gradient and momentum as approximations to regularised update descent, 2016.

[5] Filippo Cacace, Mattia Mattioni, Salvatore Monaco, and Dorothée Normand-Cyrot. A new distributed protocol for consensus of discrete-time systems. *European Journal of Control*, 74:100833, 2023.

[6] S. Cai, Z. Mao, and Z Wang. Physics-informed neural networks (pinns) for fluid mechanics: a review. *Acta Mechanica Sinica*, 37(10):1727–1738, 2021.

[7] Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *ACM Computing Surveys (CSUR)*, 39:35–53, 2007.

[8] European Commission. Protecting critical infrastructure in the eu - new rules, 2020. Accessed: 2024-05-24.

[9] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 1285–1298, New York, NY, USA, 2017. ACM.

[10] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *ICLR 2019*, 2018.

[11] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557*, 2017.

[12] Alessandro Giuseppi, Lucrezia Della Torre, Danilo Menegatti, and Antonio Pietrabissa. Adafed: Performance-based adaptive federated learning. In *2021*

The 5th International Conference on Advances in Artificial Intelligence (ICAAI), ICAAI 2021. ACM, November 2021.

[13] Alessandro Giuseppi, Sabato Manfredi, and Antonio Pietrabissa. A weighted average consensus approach for decentralized federated learning. *Machine Intelligence Research*, 19(4):319–330, 2022.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016.

[15] Yalan Guo, Yulei Wu, Yanchao Zhu, Bingqiang Yang, and Chunjing Han. Anomaly detection using distributed log data: A lightweight federated learning approach. In *2021 International Joint Conference on Neural Networks (IJCNN).* IEEE, July 2021.

[16] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction, 2018.

[17] Simon Hawkins, Hongxing He, Graham Williams, and Rohan Baxter. *Outlier Detection Using Replicator Neural Networks*, page 170–180. Springer Berlin Heidelberg, 2002.

[18] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. Drain: An online log parsing approach with fixed depth tree. *IEEE*, 2017.

[19] Wenjie Hu, Yihua Liao, and V. Rao Vemuri. Robust anomaly detection using support vector machines. In *Applications of Data Mining in Computer Security*, 2003.

[20] J. Jithish, Bithin Alangot, Nagarajan Mahalingam, and Kiat Seng Yeo. Distributed anomaly detection in smart grids: A federated learning-based approach. *IEEE Access*, 11:7157–7179, 2023.

[21] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hervé Jégou, and Tomas Mikolov. Fasttext.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651*, 2016.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2014.

[23] C. Kruegel, D. Mutz, W. Robertson, and F. Valeur. Bayesian event classification for intrusion detection. In *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE, 2010.

[24] Christopher Krügel, Thomas Toth, and Engin Kirda. Service specific anomaly detection for network intrusion detection. In *Proceedings of the 2002 ACM symposium on Applied computing*, SAC02. ACM, March 2002.

[25] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[26] Colin Lea, René Vidal, Austin Reiter, and Gregory D. Hager. *Temporal Convolutional Networks: A Unified Approach to Action Segmentation*, page 47–54. Springer International Publishing, 2016.

[27] Beibei Li, Shang Ma, Ruilong Deng, Kim-Kwang Raymond Choo, and Jin Yang. Federated anomaly detection on system logs for the internet of things: A customizable and communication-efficient approach. *IEEE Transactions on Network and Service Management*, 19(2):1705–1716, June 2022.

[28] Tian Li, Anit Kumar Sahu, Ameet Talwalkar, and Virginia Smith. Federated learning: Challenges, methods, and future directions. *IEEE Signal Processing Magazine*, 37(3):50–60, 2020.

[29] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks, 2018.

[30] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data, 2019.

[31] Xinyi Liu, Wei Liu, Xianfeng Di, Jinyue Li, Bin Cai, Wei Ren, and Hong Yang. Lognads: Network anomaly detection scheme based on semantic representation. *Future Generation Computer Systems*, 124:390–405, 2021.

[32] Jian-Guang Lou, Qiang Fu, Shengqi Yang, Jiang Li, and Bin Wu. Mining program workflow from interleaved traces. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 613–622, 2010.

[33] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[34] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 1255–1264, New York, NY, USA, 2009. ACM.

[35] Adetokunbo Makanju, A. Nur Zincir-Heywood, and Evangelos E Milios. A lightweight algorithm for message type extraction in system application logs. *IEEE Transactions on Knowledge and Data Engineering*, 24(11):1921–1936, 2012.

[36] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. *Proceedings of the 20 th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017. JMLR: W&CP volume 54*, 2016.

[37] Du Min and Li Feifei. Spell: Streaming parsing of system event logs. *IEEE*, 2016.

[38] George S. Misyris, Andreas Venzke, and Spyros Chatzivasileiadis. Physics-informed neural networks for power systems. In *2020 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5, 2020.

[39] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. In *Cryptography Mailing list at metzdowd.com*, 2008.

[40] Mirko Nardi, Lorenzo Valerio, and Andrea Passarella. Anomaly detection through unsupervised federated learning, 2022.

[41] Dinh C. Nguyen, Ming Ding, Pubudu N. Pathirana, Aruna Seneviratne, Jun Li, and H. Vincent Poor. Federated learning for internet of things: A comprehensive survey. *IEEE Communications Surveys; Tutorials*, 23(3):1622–1658, 2021.

[42] R. Olfati-Saber and R.M. Murray. Consensus problems in networks of agents with switching topology and time-delays. *IEEE Transactions on Automatic Control*, 49(9):1520–1533, 2004.

[43] Diego Ongaro and John K. Ousterhout. In search of an understandable consensus algorithm (extended version). In *USENIX Annual Technical Conference (ATC)*, pages 305–319, 2014.

[44] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[45] G Poojitha, K Naveen Kumar, and P Jayarami Reddy. Intrusion detection using artificial neural network. In *2010 Second International conference on Computing, Communication and Networking Technologies*. IEEE, July 2010.

[46] Davy Preuveneers, Vera Rimmer, Ilias Tsingenopoulos, Jan Spooren, Wouter Joosen, and Elisabeth Ilie-Zudor. Chained anomaly detection models for federated learning: An intrusion detection case study. *Applied Sciences*, 8(12):2663, December 2018.

[47] Maziar Raissi, Paris Perdikaris, and George E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[48] V. Raunak. Effective dimensionality reduction for word embeddings. In *Proceedings of the Workshop on Representation Learning for NLP (RepL4NLP) at ACL*, 2019.

[49] S P Salman and X Liu. Overfitting mechanism and avoidance in deep neural networks. *ArXiv*, 2019.

[50] Gamal Eldin I. Selim, EZZ El-Din Hemdan, Ahmed M. Shehata, and Nawal A. El-Fishawy. Anomaly events classification and detection system in critical industrial internet of things infrastructure using machine learning algorithms. *Multimedia Tools and Applications*, 80(8):12619–12640, January 2021.

[51] Weisong Shi, Jie Cao, Quan Zhang, Youhu Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[52] Marc Vucovich, Amogh Tarcar, Penjo Rebelo, Narendra Gade, Ruchi Porwal, Abdul Rahman, Christopher Redino, Kevin Choi, Dhruv Nandakumar, Robert Schiller, Edward Bowen, Alex West, Sanmitra Bhattacharya, and Balaji Veeramani. Anomaly detection via federated learning, 2022.

[53] Xiaoding Wang, Wenxin Liu, Hui Lin, Jia Hu, Kuljeet Kaur, and M. Shamim Hossain. Ai-empowered trajectory anomaly detection for intelligent transportation systems: A hierarchical federated learning approach. *IEEE Transactions on Intelligent Transportation Systems*, 24(4):4631–4640, April 2023.

[54] Wei Xu, Li Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP '09)*, pages 117–132, New York, NY, USA, October 2009. ACM.

[55] Nong Ye and Qiang Chen. An anomaly detection technique based on a chi-square statistic for detecting intrusions into information systems. *Quality and Reliability Engineering International*, 17(2):105–112, March 2001.

[56] Xiao Yu, Pallavi Joshi, Jianwu Xu, Guoliang Jin, Hui Zhang, and Guofei Jiang. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. In *Proceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 489–502, 2016.

[57] Yupeng Zhang, Lingjie Duan, and Ngai-Man Cheung. Accelerating federated learning on non-iid data against stragglers. In *2022 IEEE International Conference on Sensing, Communication, and Networking (SECON Workshops)*. IEEE, September 2022.

[58] Kanghua Zheng, Huijin Fan, Lei Liu, and Zhongtao Cheng. Triggered finite-time consensus of first-order multi-agent systems with input saturation. *IET Control Theory Appl.*, 16:464–474, 2022.

[59] Wang Zumin, Tian Jiyu, Fang Hui, Chen Liming, and Qin Jing. Lightlog: A lightweight temporal convolutional network for log anomaly detection on the edge. *Elsevier*, 203, 2022.

Ringraziamenti

Ora che sono arrivato al termine del mio percorso accademico, vorrei ringraziare tutti coloro che hanno fatto parte della mia vita in questi anni ricchi di soddisfazioni. Scelgo di scrivere questi ringraziamenti in italiano, la mia lingua nativa, perché è con essa che riesco a esprimere al meglio le mie emozioni. Il percorso che mi ha portato fin qui è composto da molti passi, ognuno dei quali è stato compiuto con una persona che mi ha accompagnato in un determinato momento, arricchendo la mia vita e, in un modo o nell'altro, aiutandomi a crescere e imparare. Tutti voi, anche se non siete menzionati, fate parte di me e del mio viaggio, che è ancora tutto da scrivere.

Prima di tutto, vorrei ringraziare il Presidente della Commissione di Laurea, Prof. Francesco Delli Priscoli, il mio relatore, Prof. Emanuele De Santis, e Co-relatore, Dr. Danilo Menegatti. Il successo di questo lavoro è stato possibile grazie al Vostri aiuto e alla Vostra pazienza. Sono particolarmente grato per l'opportunità datami di presentare la mia ricerca alla 19^a Conferenza Internazionale sulla Sicurezza delle Infrastrutture Critiche (CRITIS), un traguardo che non sarebbe stato raggiungibile senza di Voi.

Vorrei inoltre esprimere la mia più profonda gratitudine alla mia famiglia, nel cui amore e sostegno incondizionati ho trovato la chiave della mia forza. Avete supportato ogni mia decisione e mi siete sempre stati accanto, anche nei momenti più bui, facendomi capire che non sono mai stato davvero solo.

Mamma e papà, Claudia e Adriano, nonostante le vostre divergenze, non ho mai dubitato un solo istante del vostro affetto nei miei confronti, non mi avete fatto mai mancare nulla. Non servono troppe parole, basta dirvi che siete le persone a cui sono più grato. Grazie infinite.

Ai miei fratelli, Cecilia, Marianna e Francesco, voglio esprimere la mia gratitudine per il supporto indiretto che ho sempre percepito. Colgo l'occasione per dire qualche parola ad ognuno di voi.

Cec, tu in particolare non solo mi hai supportato, ma anche sopportato! Come fratello maggiore mi sento di dirti di continuare per la tua strada esattamente come stai facendo, di criticoni infatti il mondo ne è pieno. Se qualcuno ha da ridire sul tuo percorso, non ascoltarlo e anzi, sfoggiagli tutta la tua conoscenza sulle rock band finché non desiste e scappa spaventato.

Meri, non mi sento nella posizione di darti consigli sulla vita dato che in confronto a me sei una vecchia decrepita, però questa è la mia Tesi e scrivo quello che mi pare. Ti sei sempre rialzata più forte di prima dopo ogni problema, quindi prendi esempio dal passato e non abbatterti neanche questa volta.

Fra, stanno per cominciare gli anni più belli della tua vita, perciò goditi l'adolescenza prima di diventare un portiere professionista, ma attenzione a non trascurare troppo lo studio. Sappi che, come vale per le altre sorellone, potrai sempre contare su di me per qualsiasi cosa, non solo come fratello ma anche come amico.

A proposito di amici, per essere stati il mio pilastro di appoggio, vi ringrazio tutti. La vostra presenza, comprensione e incoraggiamento mi hanno aiutato enormemente a superare quasi tutti gli ostacoli lungo il mio cammino, e ce ne sono stati parecchi! Vorrei ringraziarvi personalmente tutti, ma siete troppi e, nel mentre che scrivo, manca pochissimo alla discussione di Laurea che devo ancora iniziare a preparare, perciò, voi che non siete stati nominati, non offendetevi perché avete lo stesso un posto nel mio cuoricino.

Comincio col ringraziare il prossimo miglior veterinario del pianeta, sempre se non verrà travolto prima dai debiti di gioco. Parlo ovviamente di te, Riccardo. Sei lontano da Mostacciano ormai da tanti anni, ma ogni volta che ci rivediamo è come se non fossi mai partito. Hai un cuore enorme, quasi come la tua fortuna nel lancio della moneta, e tutti lo notiamo.

Rimanendo in zona, mi sembra il minimo menzionare Lorenzo, il mio caro compagno di passeggiate notturne e gite senza meta. È sempre un piacere trascorrere il tempo con te a rievocare vecchi episodi del nostro passato – chissà quante altre storie ancora dobbiamo vivere e raccontarci – tranne quando mi tieni in ostaggio nella mia stessa macchina perché non vuoi salire a casa.

Per Valerio invece è il contrario, devo aspettare mezz'ora sotto casa tua prima di vederti scendere. Nonostante ciò sono veramente felice di averti di nuovo presente nella mia vita proprio come un tempo, quando eravamo due cretini infantili (non che sia cambiato molto adesso). Sei la persona più artistica che conosca: fotografo, chitarrista, ma soprattutto cuoco. Quando mi passi la ricetta della Coleslaw?

Rico, Ric, Victor, decidi tu come vuoi essere chiamato. Tu sei quell'amico a cui ho capito di poter dire tutto, sai ascoltare e dare i giusti consigli motivazionali. Se mai combinerò qualche guaio con la legge sarai il primo a cui mi rivolgerò, come d'altronde faresti anche tu con me. Non dimenticherò mai il giorno che te e Guddy mi avete trascinato al parco per fare i primi DIP. Sbrigati a tornare, mi manchi un sacco.

Pier, ti conosco solo da un anno ma il legame che ci unisce è già molto forte. Formiamo un bel duo anche se con Rico diamo il meglio di noi. Grazie per tutti i consigli che anche tu hai saputo darmi, ora testa alle prossime avventure che ci aspettano.

Caro Guglielmo, tra noi c'è un altro tipo di amicizia, abbiamo un rapporto di amore ed odio, o meglio, di tenda ed astice. A me piace così. Dato che non posso elencare tutti i tuoi pregi perché verrebbe una Tesi di duemila pagine, vorrei dirti qualcosa di divertente, che confermi un'altra volta il mio umorismo sopraffino. Ma non mi viene nulla in mente, perciò mi limito a dedicarti questa immagine:

Dopo la triennale abbiamo preso strade accademiche diverse, ma siamo rimasti lo stesso tanto uniti. Lorenzo, ti devo ringraziare di cuore per la tua guida, non solo nella scelta degli esami liberi – che ci ha consentito di vederci periodicamente oltre che alle settimanali partite di calcetto al pizzicotto – ma anche nella vita, tirandomi su nei miei momenti più bui con la tua comprensione ed empatia.

Ci terrei a ringraziare anche Edoardo, il mio gym bro universitario. Sei un gigante dal cuore d'oro, non ti ho mai visto ancora arrabbiato, e per questo mi stupisco tutte le volte quando mi racconti la storia della sedia lanciata! Anche se la palestra è la nostra costante, mancheranno i giorni in Università.

Mancherà vedere in Università anche tutti "quelli della mensa". Mi riferisco a Ciccio, Lorenzo, Antonio, Antonio, Antonio (non sono impazzito, sono 3 persone diverse!), Dumitru, il duo Alessia e Michela, Kiro che si laurea con me oggi, obviously my big turk friend Mert, e tutti gli altri! Grazie infinite per le ore passate in aula A3 a giocare a briscola e scopone, tra una pausa caffè e l'altra. Senza di voi avrei sicuramente studiato di più, ma che noia sarebbe stata!