Università degli Studi di Ferrara

DOCTORAL COURSE IN

ENGINEERING SCIENCE

CYCLE XXXVIII

COORDINATOR Prof. Trillo Stefano

Enabling Machine Learning Services in High-Stakes Environments

Scientific/Disciplinary Sector (SDS) ING-INF/05

Candidate

Dott. Simon Dahdal

Supervisor

Prof. Mauro Tortonesi

Year 2022/2025

Acknowledgements

This thesis marks the conclusion of my Ph.D. journey at the University of Ferrara, and it would not have been possible without the help and support of many people.

Special thanks to Prof. Mauro Tortonesi from the Department of Mathematics and Computer Science and to Prof. Cesare Stefanelli from the Department of Engineering at the University of Ferrara for their guidance and mentorship throughout this work.

Special thanks to Dr. Niranjan Suri from the Florida Institute for Human and Machine Cognition (IHMC) for his assistance during my Ph.D. and throughout my internship, which was an invaluable experience.

Finally, I want to thank my family and friends. Their steady encouragement, understanding, and patience have been important throughout this journey.

Abstract

Machine Learning is a transformative technology whose ability to extract patterns from complex data has enabled the development of novel applications, including predictive analytics, adaptive control, and intelligent decision-making across diverse domains. We define High-Stakes Environments as operational contexts where errors or failures carry disproportionate operational, economic, or human consequences. In these settings, success depends not only on the predictive accuracy of the ML models but also on the reliability of entire systems under the infrastructural uncertainty, contextual variability, and organizational constraints. Managing the machine learning life cycle – from data collection to training, and deployment is especially demanding in these contexts, underscoring the need for systematic guidance to enable ML in a resilient, efficient, and context-aware manner.

This work starts by addressing this need by introducing a framework for categorizing High-Stakes Environments, providing a structured entry point to start applying ML. The typology framework classifies operational settings along three key connectivity network dimensions–reliability, bandwidth, and latency–and links them directly to implications for ML workflows, particularly regarding training and deployment, highlighting how connectivity constraints affect the engineering choices and strategies required in such contexts. The framework distinguishes three macro-types of High-Stakes Environments: Type 1 corresponds to well-connected and resource-rich environments, where stable connectivity, abundant computational capacity, and predictable conditions allow ML straightforward deployment. Type 2 represents partially connected and resource-limited environments, where constraints emerge that require explicit trade-offs between centralized training and local inference, demanding ML optimization and hybrid strategies. Type 3 represents severely degraded or entirely disconnected infrastructures, typically encountered in extreme conditions such as natural or human-made disaster zones.

In such contexts, systems must operate autonomously under unstable circumstances. Consequently, algorithmic adaptability becomes critical to preserving resilience.

The work follows up with three representative real-world case studies. A smart manufacturing environment exemplifies Type 1 settings, where stable settings act as enablers for large-scale ML integration. This case provides a concrete end-to-end use case, involving the development of ML applications for anomaly detection and gearbox classification, and the final deployment of the solution in production. An in-field industrial scenario reflects Type 2 environments, demonstrating how ML workflows must respect the constraints at hand. A complete system was designed to make the company's ice cream machines self-adaptive by classifying errors in the used recipes and consequently adjusting processing parameters in real time to preserve the product quality. Finally, a disaster-impacted environment captures the essence of Type 3 settings, highlighting the necessity of new adaptive ML approaches that sustain autonomy under degraded conditions. Thereby, a new distributed continual learning framework, Roaming Machine Learning (RoamML), was developed. It relies on intelligent mobile agents that navigate dynamically across network nodes, continuously learning from newly available data. Guided by a Data Gravity strategy, the agents make optimal, context-aware choices about the next node to visit, ensuring resilient and timely local decision-making in highly unstable environments.

The research presented in this dissertation has been conducted in close collaboration with international institutes and world-leading manufacturing industries, including Bonfiglioli Riduttori and Carpigiani, and was further enriched by a research period at the Florida Institute for Human and Machine Cognition (IHMC), Florida, USA.

Sommario

Il Machine Learning si è affermato come tecnologia trasformativa, capace di estrarre dai dati dei pattern complessi e supportare applicazioni innovative quali analisi predittive, controllo adattivo e decisioni intelligenti in diversi domini. Definiamo gli High-Stakes Environments come contesti operativi caratterizzati da sfide, in cui errori o malfunzionamenti generano gravi conseguenze operative, economiche o per l'incolumità delle persone. In tali scenari, l'efficacia non dipende soltanto dall'accuratezza predittiva dei modelli di ML, ma anche dalla resilienza dell'intero sistema in condizioni di incertezza a livello infrastrutturale, variabilità contestuale e vincoli organizzativi. La gestione del ciclo di vita del ML (dalla raccolta dei dati al deployment) in tali scenari è complessa e mette in evidenza la necessità di linee guida per soluzioni resilienti ed efficienti.

Questo lavoro introduce un framework per categorizzare tali contesti, offrendo un approccio sistematico all'applicazione del ML. Il framework classifica i contesti secondo tre caratteristiche chiave della connettività – affidabilità, banda disponibile e latenza – e collega tali caratteristiche alle implicazioni sui flussi di lavoro di ML, in particolare per quanto riguarda le fasi di training e deployment. In questo modo viene mostrato come i vincoli di connettività condizionino le scelte ingegneristiche e le strategie necessarie in tali scenari. Vengono definite tre tipologie principali: la prima riguarda ambienti ben connessi e ricchi di risorse, nei quali connettività stabile, risorse computazionali abbondanti e condizioni operative prevedibili consentono un deployment relativamente diretto. La seconda include ambienti parzialmente connessi e con risorse limitate, in cui emergono vincoli che impongono compromessi tra addestramento centralizzato e inferenza locale, richiedendo l'ottimizzazione dei modelli e strategie ibride. La terza comprende infrastrutture gravemente degradate o totalmente disconnesse, tipiche di scenari estremi come disastri di origine naturale o umana, nelle quali i sistemi devono operare in autonomia in condizioni instabili, rendendo l'adattabilità algoritmica fondamentale per preser-

vare la resilienza.

Il lavoro prosegue presentando tre casi d'uso reali, ciascuno esemplificativo di un diverso macrotipo del framework proposto. Si inizia con un ambiente di smart manufacturing, rappresentativo del Tipo 1, nel quale la stabilità infrastrutturale funge da abilitatore per l'integrazione del ML su larga scala. Sviluppando applicazioni di ML per il rilevamento di anomalie e la classificazione di riduttori malfunzionanti, fino al deployment in produzione. Successivamente, viene affrontato uno scenario industriale riconducibile al Tipo 2, dimostrando l'adattamento ai vincoli di connettività e risorse. In questo caso è stato progettato un sistema completo che rende le macchine per gelato auto-adattive, classificando automaticamente gli errori nelle ricette e modificando in tempo reale i parametri di processo, al fine di preservare la qualità del prodotto. Infine, un ambiente di disastro cattura l'essenza dei contesti di Tipo 3, mettendo in evidenza la necessità di nuovi approcci adattivi al ML in grado di garantire autonomia in condizioni degradate. A tal fine è stato sviluppato un nuovo framework di Distributed Continual Learning – Roaming Machine Learning (RoamML) – basato su agenti mobili intelligenti che navigano dinamicamente tra i nodi di rete, guidato da una strategia basata sul data gravity, consentendo di scegliere in maniera ottimale il nodo successivo da visitare e apprendendo continuamente dai dati.

La ricerca presentata in questa tesi è stata condotta in stretta collaborazione con istituti internazionali e aziende manifatturiere di livello mondiale, tra cui Bonfiglioli Riduttori e Carpigiani, ed è stata ulteriormente arricchita da un periodo di ricerca presso il Florida Institute for Human and Machine Cognition (IHMC), Florida, USA.

List of Acronyms

AD Anomaly Detection 73–75, 78, 79, 81, 82, 89

AE autoencoder 79–81, 89

AHP Analytical Hierarchy Process 40–42

AI Artificial Intelligence 16

ANN Artificial Neural Networks 30

APIs application programming interfaces 27

BWM Best Worst Method 41, 139, 140

C2 Command and Control 16, 49

CL Continual Learning 36, 111, 112, 114, 126, 129, 131

CNN Convolutional Neural Networks 31, 98

CTGAN Conditional Tabular GAN 32, 76

DDIL Disrupted, Degraded, Intermittent, and Low-Bandwidth 7, 61, 62, 66, 127, 136

DG Data Gravity iii, 7, 116–119, 125, 126, 131–134, 136, 139, 142

DGM Deep Generative Models 31

ER Experience Replay 36, 115, 127

FC Fully Connected 99, 102

FL Federated Learning 34, 51, 110, 111, 133

GAN Generative Adversarial Networks 31

GRU Gated Recurrent Unit 98

HADR Human Assistance & Disaster Recovery 14, 15

HADR Human Assistance & Disaster Recovery 111, 118, 123, 132, 139

HOT Hard-O-Tronic 94

HOT-AI Hard-O-Tronic AI-driven 94, 97

HS High-Stakes 8, 69, 89

HSE High-Stakes Environments ii, 5, 6, 8, 9, 13, 14, 17–19, 21–23, 49–52, 54, 56, 70, 88, 89, 93, 107, 109, 151–153, 193

IoT Internet of Things 14, 16

LC life cycle 26, 47, 49, 50, 59, 89

LIC Least Important Criteria vs. other criteria 137, 139

LogReg Logistic Regression 29, 82, 89

LSTM Long short-Term memory 98

MADM Multi-Attribute Decision Making 39, 41

MCDM Multi-Criteria Decision Making 9, 39, 42, 43, 45, 119, 125, 126, 140, 142

MIC Most Important Criteria vs. other criteria 137, 139

ML Machine Learning 14–18, 20, 22, 23, 25–32, 34, 36–38, 43–45, 47–51, 54, 58, 59, 61–64, 68–70, 72, 73, 78–81, 83, 84, 89, 91, 93, 110

ML Machine Learning 114, 116

MLLC machine learning life cycle ii, 5, 8, 9, 25, 42, 45, 83, 87, 152

MLOps Machine Learning Operations 105

MLP Multilayer Perceptron 30

MTS multivariate time series 73, 74, 79, 80, 89, 96, 107

PCA Principal Component Analysis 29

RNN Recurrent Neural Networks 31, 98

RoamML Roaming Machine Learning iii, vi, 7, 112, 114–119, 123, 125–133, 136, 138–144, 190, 191, 194

SPP Spatial Pyramid Pooling 102, 103

SVM Support Vector Machines 29

TOPSIS Technique for Order of Preference by Similarity to Ideal Solution 42, 137, 139, 140, 142

TVAE Tabular Variational Autoencoder 32

UAV unmanned aerial vehicles 16, 49, 123

VAE Variational Autoencoders 31

WGAN Wasserstein Generative Adversarial Network 32, 76

Contents

Chapter 1

Introduction

Machine Learning has established itself as a cornerstone technology, profoundly reshaping a wide spectrum of domains. By enabling predictive analytics, adaptive control, and intelligent decision-making at scale, ML has unlocked capabilities that were previously unattainable. Its strength lies in the ability to uncover latent patterns within complex, high-dimensional datasets to support dynamic data-driven processes, thereby fostering the development of novel applications across scientific and industrial fields [1]–[3].

In our prior work [4], we define High-Stakes Environments as operational contexts that present distinct challenges. In these environments, system errors or failures can lead to disproportionately severe consequences – whether operational, economic, or human in nature. The deployment of ML in such settings moves the discussion beyond questions of algorithmic performance, raising concerns of reliability, robustness, and adaptability under real-world constraints. Here, success is measured not only by predictive accuracy but also by the ability of ML systems to maintain functionality amid infrastructural uncertainty, contextual variability, and organizational limitations.

Effectively managing the entire machine learning life cycle, encompassing data col-

lection, preprocessing, model selection, training, and deployment, becomes uniquely demanding in High-Stakes Environments. Technical constraints intersect with organizational and environmental factors, directly influencing feasibility, resilience, and trustworthiness [5]. This convergence underscores the necessity for systematic frameworks and clear methodological guidance to ensure that ML solutions can be engineered and deployed in ways that are robust, efficient, and acutely aware of the contextual realities of high-stakes operations [6].

This dissertation addresses these challenges by introducing a typology framework for categorizing High-Stakes Environments, offering engineers a structured foundation for applying ML and making context-aware design decisions. The framework classifies operational settings based on three key connectivity characteristics: reliability, bandwidth availability, and latency. These characteristics are directly linked to their implications for ML workflows, shaping how data is collected, how models are trained, and how inference is deployed across different infrastructural contexts. By systematically connecting connectivity characteristics to ML requirements, the framework demonstrates how technical constraints guide engineering decisions and determine the strategies necessary for resilient, efficient, and scalable deployments. It distinguishes three macro-types of High-Stakes Environments, each defined by specific enablers, limitations, and design imperatives. The first type corresponds to well-connected and resource-rich environments, where stable connectivity, abundant computational resources, and predictable operating conditions provide ideal conditions for large-scale ML adoption. In such settings, centralized training and real-time deployment can be implemented straightforwardly with minimal compromise. The second type captures partially connected and resource-constrained environments. Here, engineers must balance centralized training and local inference, as connectivity interruptions and hardware limitations impose significant constraints. These contexts demand optimization techniques – such as model compression, pruning, and hybrid cloud–edge deployment – to achieve an appropriate

trade-off between performance, efficiency, and resource availability. The third type represents severely degraded or entirely disconnected infrastructures, typically arising in extreme conditions such as natural or human-made disasters. In these scenarios, ML systems must operate autonomously under unstable and unpredictable circumstances. Success in such contexts requires algorithmic adaptability and innovative workflow designs capable of ensuring resilience and sustaining autonomy in the absence of reliable infrastructure.

The work follows up with three representative real-world case studies, thereby grounding the typology framework in practice and illustrating how each scenario can be systematically approached. A smart manufacturing environment at Bonfiglioli Riduttori exemplifies Type 1 settings, where stable connectivity, abundant computational resources, and predictable operating conditions act as enablers for large-scale ML integration. This case provides a concrete end-to-end use case, involving the development of ML applications for anomaly detection and gearbox classification, and the final deployment of the solution in production. An in-field industrial scenario at Carpigiani reflects Type 2 environments, demonstrating how ML workflows must adapt to connectivity limitations and hardware constraints through model optimization and hybrid cloud–edge solutions. A complete system was designed to make the company's ice cream and gelato machines self-adaptive by automatically classifying errors in the mixture recipes and consequently adjusting processing parameters in real time, thereby preserving product quality and preventing batch losses. Finally, a disaster-impacted environment captures the essence of Type 3 settings, highlighting the necessity of adaptive ML approaches that sustain autonomy under Disrupted, Degraded, Intermittent, and Low-Bandwidth conditions. To address this challenge, a newly developed distributed continual learning framework, Roaming Machine Learning (RoamML), was developed. The framework is based on intelligent mobile agents that navigate dynamically across network nodes, continuously learning from newly available data. Navigation decisions are guided by a Data Gravity

strategy, which enables agents to make optimal, context-aware choices about the next node to visit, thereby supporting resilient and timely local decision-making in highly unstable environments.

To guide the investigation presented in this dissertation, a central research hypothesis is formulated: that the effectiveness, robustness, and operational feasibility of Machine Learning systems in High-Stakes Environments are fundamentally shaped by infrastructure connectivity characteristics – specifically reliability, bandwidth, and latency – and that a systematic typology based on these characteristics can provide a structured foundation for designing robust and context-appropriate ML workflows. Building on this hypothesis, the research addresses several key questions: how connectivity constraints influence the different phases of the machine learning life cycle; whether High-Stakes Environments can be meaningfully classified according to connectivity conditions to support engineering decision-making; how ML training and deployment strategies must be adapted across varying infrastructure scenarios; and whether adaptive and distributed approaches can improve learning effectiveness and system resilience under degraded or disconnected conditions. These questions are investigated through the development of a connectivity-aware typology framework and its validation across representative real-world case studies, spanning well-connected industrial environments, resource-constrained in-field systems, and disaster-impacted scenarios.

This thesis is organized as follows:

Chapter 2 introduces the notion of High-Stakes Environments, beginning with their definition and two representative domains: industrial manufacturing and disaster response and recovery. It then explores the core characteristics that distinguish these contexts, focusing on the irreversibility and severity of consequences as well as the real-time demands placed on operations. The discussion then turns to network connectivity, examining its defining characteristics of reliability, bandwidth, and latency, and how they influence the behavior of systems in High-Stakes settings.

Chapter 3 provides the foundational background and reviews related work that underpin the rest of the thesis. The chapter begins by outlining the machine learning life cycle, presenting its main phases from data collection and processing to deployment and monitoring, with emphasis on the challenges and requirements at each step: data management, model selection, training strategies, and deployment. It then introduces Multi-Criteria Decision Making methods as tools to support the evaluation, prioritization, and selection of ML strategies in complex environments, covering criteria weighting, alternative selection, and their role in guiding decisions across the life cycle. Finally, the chapter reviews key challenges in applying ML, grouping them into data-related, model-related, and infrastructural categories.

Chapter 4 examines the role of Machine Learning in High-Stakes Environments. It begins by surveying the range of applications in which ML technologies are increasingly adopted, before analyzing the specific implications these contexts impose on ML systems. Particular attention is given to the constraints that affect both training and deployment across different operational settings. Building on this foundation, the chapter introduces a typology framework that organizes High-Stakes Environments according to their connectivity conditions and corresponding ML workflows, considering cloud-based, edge-based, and hybrid training–deployment strategies. The discussion then turns to three representative archetypes: smart manufacturing environments, which highlight the enabling role of robust infrastructure; in-field industrial environments, where constraints emerge that require balancing between cloud and edge; and disaster-impacted environments, where autonomy under degraded infrastructure makes adaptability essential. The chapter also provides practical guidelines for classifying High-Stakes Environments, offering readers a structured entry point for analyzing requirements and designing resilient ML solutions.

Chapter 5 presents the first representative case study, situated in a smart manufacturing environment at Bonfiglioli Riduttori. It begins with an overview of two production

lines, a gear hobbing machine and a gearbox assembly plant, and outlines the specific goals and requirements driving the adoption of ML in these settings. The discussion then turns to dataset collection and preprocessing, essential foundations for model development. Building on this, the chapter details the design and training of ML models for anomaly detection, including the development of classifiers tailored to different process stages. Attention is also given to MLOps platform deployment, highlighting the integration of solutions into the Bi-Rex and NGA4M platforms to ensure scalable and maintainable operation.

Chapter 6 introduces the second case study, focusing on in-field industrial environments through the use case of Carpigiani. It begins by outlining the company context, along with the goals and requirements that motivated the integration of ML solutions into their production systems. The chapter then presents the HoT-AI system, describing its architecture, system design, and the development of a multi-milestone classifier tailored to the specific challenges of in-field operations. Following this, the process of dataset collection and preprocessing is detailed and was prepared for model development. The discussion then moves to model design and training, highlighting the iterative training of the multi-milestone classifier through progressive fine-tuning and highlighting the trade-offs required to balance performance with resource constraints. Finally, the chapter illustrates the deployment of both the model and the supporting platform, demonstrating how hybrid cloud–edge solutions can achieve real-time adaptability despite connectivity and hardware limitations.

Chapter 7 presents the third case study, situated in disaster-impacted environments where infrastructures are severely degraded or disconnected. It begins by rethinking distributed training, motivating the need for new approaches under degraded, contested, or delay-tolerant conditions. The chapter then introduces the RoamML framework, detailing its architectural design and platform implementation as a novel distributed continual learning system. Building on this, the concept of data gravity is explored, presenting

both the underlying model and the use of multi-criteria decision-making to guide mobile agents in selecting the optimal path across network nodes. The testing methodology is then described, including the experimental configuration and performance metrics used to evaluate the framework. Finally, the chapter reports on a comprehensive experimental evaluation, comparing centralized, federated, and RoamML-based approaches under different scenarios, including single-criteria, context-aware, and context-independent settings.

Chapter 9 provides conclusive remarks and discusses some future works.

Chapter 2

High-Stakes Environments

High-Stakes Environments are operational domains in which the performance, reliability, and robustness of a system have immediate and significant real-world consequences. Examples include industrial manufacturing, disaster response, and post-crisis recovery, where failures can result in substantial economic loss, safety hazards, or delays in critical decision-making.

2.1 Definition of High-Stakes

High-Stakes Environments (HSE) are settings in which the cost of system failure or suboptimal performance is exceptionally high, often involving safety-critical, mission-critical, or economically significant outcomes. These environments are characterized by high-complexity, rapidly-changing conditions, and tightly coupled, interdependent components operating across multiple technological and organizational layers [4]. Such settings typically involve the convergence of technological infrastructure, logistical constraints, environmental variability, and human decision-making, all of which interact in

Figure 2.1: Example of High-Stakes Scenarios: Illustrations include advanced industrial automation, robotic manufacturing, autonomous robotic assembly lines, natural disasters via extreme weather events such as tornadoes, climate-driven wildfires, severe winter storms, and human-made disasters such as post-conflict urban and infrastructure destruction.

real time. The presence of uncertainty, time pressure, and incomplete information further amplifies the need for robust, adaptive, and interpretable ML systems capable of supporting high-consequence decision-making [7]. While the concept of High-Stakes Environments (HSE) spans a wide range of domains, including healthcare, defense, and autonomous systems, this work focuses on two particularly critical areas: *industrial manufacturing* [8] and *Human Assistance & Disaster Recovery (HADR)* operations [9], [10]. These domains exemplify the challenges of deploying AI systems in scenarios where reliability, responsiveness, and situational awareness are paramount.

2.1.1 Industrial Manufacturing Scenarios

The transition toward Industry 5.0 goes beyond the fourth industrial revolution, which primarily focused on interconnecting cyber-physical systems. Industry 5.0 represents a paradigm shift in manufacturing, where technological innovation is aligned with human-centric values, sustainability, and the resilience of production ecosystems [3]. This evolution is driven by the advancement of enabling technologies such as the Internet

of Things, Machine Learning, and Big Data analytics, which together foster adaptive, intelligent, and sustainable manufacturing systems. At the core of this transformation are data-driven services that facilitate human-machine collaboration, enhance resilience across operational layers, and promote resource-efficient production processes. ML algorithms play a central role in achieving these objectives by enabling real-time decision-making, predictive maintenance, and intelligent automation.

Recent literature highlights the shift from traditional automation paradigms toward AI-driven systems that are not only efficient but also context-aware and human-centric [6]. These developments reflect a broader redefinition of value in industrial systems, where adaptability, safety, sustainability, and productivity are equally prioritized. Unlike earlier industrial revolutions that focused primarily on maximizing throughput and minimizing cost, Industry 5.0 places greater emphasis on robustness, resilience, and the seamless integration of human expertise into AI-driven workflows. This paradigm shift encourages the design of socio-technical systems where human judgment and machine intelligence co-evolve to meet complex operational demands [2] .

2.1.2 Disaster Response and Recovery Scenarios

Human Assistance & Disaster Recovery refers to the coordinated efforts to provide urgent support to populations affected by natural or man-made disasters [9]. These efforts are aimed at minimizing human suffering, reducing casualties, and restoring stability to disaster-impacted regions. To achieve these objectives effectively, HADR operations rely on timely information, robust coordination among multiple stakeholders, and the rapid deployment of both human and technological resources [11]. Disaster management is typically framed around four core stages: preparedness, response, recovery, and mitigation/prevention [10], [12]. Yu in [11] expands the framework into more detailed macro-phases, covering tasks such as damage assessment, post-disaster coordination, long-term recovery and mitigation, continuous monitoring, and predictive analysis. Each

phase presents distinct operational requirements, often under conditions of limited time, scarce resources, and degraded infrastructure.

Additionally, recent literature [10] highlights significant advancements in applying AI to support decision-making throughout this cycle. Applications include early warning systems for seismic activity, post-disaster damage assessment using satellite imagery, and optimization of logistics in conflict zones. These developments underscore the critical role of intelligent systems and ML applications in enhancing situational awareness, coordination, and response effectiveness in Disaster Response and Recovery [13]–[16].

A critical enabler of effective disaster response lies in the integration of diverse data sources. Yu's seminal work provides a comprehensive account of data types relevant to natural disaster management [11], including satellite imagery, sensor network outputs, IoT streams, unmanned aerial vehicles-acquired data, LiDAR scans, geospatial records, simulation outputs, and user-generated content such as social media and crowd-sourced information. Complementing this perspective, [5] details the data types and nodes available in tactical scenarios and emphasizes the role of data, IoT, and AI in Command and Control (C2) operations. C2 is typically structured into six fundamental functions: observation, analysis, decision, planning, execution, and assessment. Therefore, the ability to fuse these heterogeneous sources and technologies into a coherent operational picture is essential for situational awareness and informed decision-making under conditions of uncertainty, both in civilian disaster relief and in tactical environments. The integration of open-source data, ranging from satellite feeds to social media streams, further strengthens real-time disaster monitoring and enhances predictive capabilities, enabling more accurate forecasting and proactive interventions.

2.2 Core Characteristics of High-Stakes Environments

High-Stakes Environments are distinguished primarily by their social and ethical implications [17]. Decisions made in these contexts often carry irreversible consequences, affect human lives and welfare, and must be executed under strict temporal and operational constraints. As such, Machine Learning systems deployed in these settings must be designed with an acute awareness of the broader impacts of automation and algorithmic decision-making.

Irreversibility and Consequence Severity: One of the defining features of High-Stakes Environments is the severity and irreversibility of potential outcomes. Errors in these contexts may lead to significant material losses, irreversible environmental damage, or risks to human safety and well-being [17]. For instance, a misclassification in a predictive maintenance system may result in critical equipment failure, while a faulty decision in disaster response may delay aid to vulnerable populations. This characteristic necessitates a high degree of model reliability, transparency, and accountability. Systems must be robust against uncertainty, capable of justifying their decisions, and designed to incorporate human oversight where necessary. The ethical imperative is not merely to optimize for performance, but to minimize harm and support human-centered values in decision-making processes.

Real-Time Operational Demands: High-Stakes Environments frequently demands swift and context-aware responses. Whether in industrial control systems or humanitarian response scenarios, decision latency must be minimized without compromising accuracy. This introduces challenges related to model inference speed, data streaming, fault tolerance, and computational efficiency, especially when systems are deployed on edge or resource-constrained hardware. Real-time constraints also amplify the importance of monitoring, fail-safes, and adaptive control mechanisms. Machine Learning applica-

tions must continuously operate under uncertain, dynamic, and noisy conditions while maintaining consistent performance. This includes handling incomplete data, reacting to system perturbations, and ensuring graceful degradation in case of partial failures. In these environments, the fusion of AI with real-time systems engineering becomes essential, not only to meet performance requirements but to ensure safe and reliable operation in mission-critical contexts.

2.3 Network Connectivity in High-Stakes Environments

A systematic assessment of data-driven applications in general, and ML-driven applications in particular, requires a clear understanding of the technical constraints that determine both their feasibility and performance within High-Stakes Environments. Among these, the most fundamental is the *computational capacity* of the nodes within the operational network. Computational capacity refers to the processing power, memory, and hardware acceleration available on the devices or nodes where Machine Learning models are trained and/or deployed. This includes CPU performance, GPU availability, RAM size, and the presence of specialized accelerators such as TPUs or NPUs. Operational environments vary significantly along this dimension, from high-performance cloud infrastructures with virtually unlimited, on-demand scalability to constrained edge devices such as embedded controllers or single-board computers with limited processing power, memory, and energy budgets. The available computational capacity directly influences multiple aspects of an ML system's design and operation: it determines the maximum complexity of data analytics and ML models that can be trained or deployed, the feasible batch sizes for inference, and the ability to perform on-device training or fine-tuning. It also affects inference latency, which is often critical for meeting the real-time or near-real-time requirements typical of High-Stakes Environments.

While computational capacity is often the primary constraint, it can usually be mit-

igated through appropriate hardware provisioning. Once adequate computing resources are in place, the dominant limitations tend to arise from network connectivity. This can be described along three key connectivity characteristics: *Reliability*, *Bandwidth*, and *Latency*. Reliability concerns the continuity of correct communication service; bandwidth determines the volume of data that can be transmitted per unit of time; and latency captures the delay between transmission and reception. Although the broader concept of dependability, as presented by Avizienis et al. [18], encompasses additional attributes such as safety, integrity, and maintainability, this work focuses specifically on communication-related aspects that directly affect distributed learning and deployment. Together, reliability, bandwidth, and latency determine how efficiently data and models can be exchanged between nodes, thereby directly influencing the selection of training paradigms and deployment strategies in High-Stakes Environments.

2.3.1 Reliability

Reliability refers to the continuity of correct service delivery in the communication links within the operational environment [18]. High network reliability is characterized by stable connections with minimal failures, enabling frequent and uninterrupted data exchange between system components, such as between edge devices and cloud infrastructure, so that communication services remain continuously correct over time. In contrast, low network reliability is associated with intermittent connectivity, high packet loss, or limited communication windows, conditions often encountered in disaster-response operations, rural areas, or certain industrial edge deployments, where the continuity of correct communication service cannot be assured. Common metrics for quantifying network reliability include packet delivery ratio, mean time between failures, and uptime percentage; however, strictly speaking, these metrics approximate the frequency and duration of service failures and thus quantify the continuity of correct service over time. In High-Stakes Environments, reliability is a decisive factor: unstable connections can

hinder real-time synchronization, delay or block model updates, and disrupt cloud-based inference. These challenges may necessitate alternative deployment strategies, such as edge inference, local decision-making, or asynchronous update mechanisms, to preserve continuity of correct service despite communication faults, ensuring operational continuity.

2.3.2 Bandwidth

Bandwidth refers to the maximum volume of data that can be transmitted across a network within a given time frame. High bandwidth enables frequent and large-scale data transfers, facilitating operations such as continuous model updates, streaming high-resolution sensor data, and performing centralized analytics. In contrast, limited bandwidth restricts both the frequency and size of data transmissions, directly impacting the feasibility of centralized training for ML systems, real-time monitoring, and full system observability. Such limitations also influence the update strategies and frequency of deployed systems. In resource-constrained environments, bandwidth limitations may arise from various causes such as contractual restrictions, available physical network infrastructure, or the inherent characteristics of the communication medium. These constraints often necessitate the use of techniques such as compression, edge-level preprocessing, and efficient data encoding to minimize transmission loads. In more extreme cases, they may require novel and ad-hoc training strategies tailored to the available connectivity. Bandwidth constraints can also drive the adoption of edge-first architectures, in which critical processing is performed locally and only essential or aggregated results are transmitted to the cloud.

2.3.3 Latency

Latency refers to the time delay between the transmission of data and the receipt of the corresponding response. In networking terms, it is often measured as round-trip time (RTT) and expressed in milliseconds. Low latency enables rapid request-response cycles, which are essential for applications requiring real-time or soft real-time decision-making, such as closed-loop control systems, industrial robotics, or autonomous navigation. Conversely, high latency introduces delays that can degrade system performance, disrupt time-sensitive operations, or render certain functionalities unusable. In High-Stakes Environments, latency is a decisive factor in determining deployment strategies. High latency can make cloud-based inference impractical for applications where immediate feedback is critical, thereby necessitating edge inference or hybrid architectures that reduce dependency on remote processing. Latency can be influenced by factors such as network congestion, physical distance between nodes, routing inefficiencies, and protocol overheads. Common techniques for mitigating latency include optimizing network routing, caching frequently accessed data locally, employing lightweight model architectures, and leveraging hardware accelerators at the edge. In extreme cases, high latency may necessitate fully decentralized processing and, where applicable, localized training to minimize data transmission hops and keep computation as close as possible to the data sources.

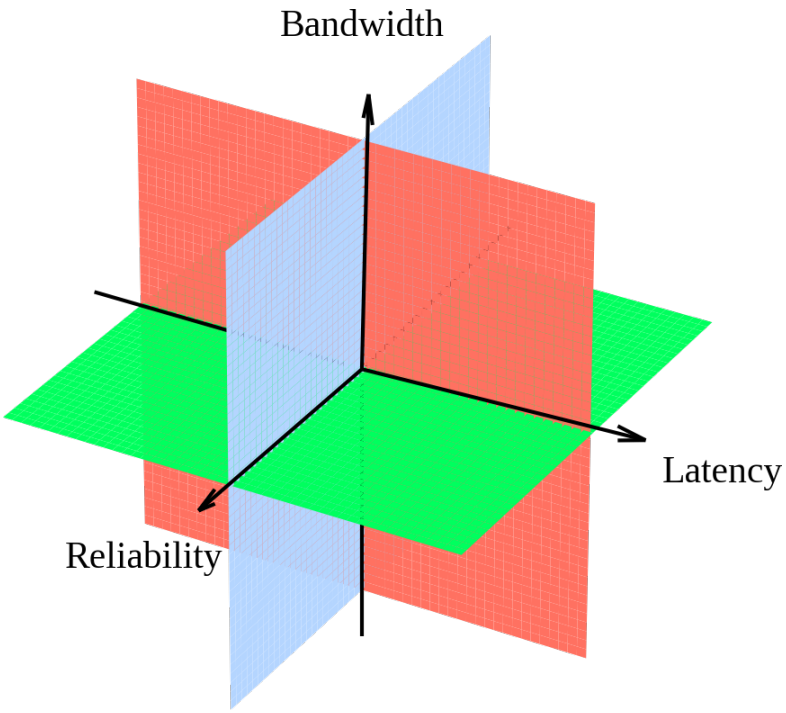


Figure 2.2: Three orthogonal dimensions characterizing network connectivity: reliability (continuity of correct communication service), bandwidth (data transfer capacity per unit time), and latency (communication delay). Their combination defines the operational communication regime that influences distributed training and deployment strategies.

2.4 Chapter Summary

This chapter examined the defining characteristics and technical constraints of High-Stakes Environments with a particular focus on their implications for the applicability of Machine Learning systems. First, the notion of high-stakes environments was contextualized through two illustrative domains: industrial manufacturing and disaster response. These scenarios highlight not only the operational complexity of such environments but also the profound human, economic, and environmental consequences that may arise from system failures. Second, the discussion outlined two core characteristics of High-Stakes Environments: irreversibility and consequence severity, as well as real-time op-

erational demands. These features distinguish high-stakes settings from conventional domains and impose strict requirements on the design, deployment, and assurance of Machine Learning-enabled systems. Finally, the chapter introduced the three primary network connectivity characteristics, *reliability*, *bandwidth*, and *latency*, that govern the feasibility and performance of distributed learning and inference in these contexts. While computational capacity can often be addressed through appropriate provisioning, network connectivity remains a persistent constraint. Each axis introduces trade-offs that shape architectural choices, influencing whether data and models are processed centrally, distributed across the edge, or deployed in hybrid configurations.

Taken together, these elements provide a systematic framework for evaluating Machine Learning in High-Stakes Environments. By recognizing both the operational characteristics and technical constraints.

Chapter 3

Machine Learning Life Cycle Management

Machine Learning Life Cycle Management refers to the comprehensive and systematic process of developing, deploying, and managing Machine Learning systems. It encompasses all critical phases, including model training, deployment, monitoring, and ongoing maintenance. Effective management also requires handling data and infrastructure to ensure that models remain aligned with the specific requirements of each environment. Fig. 3.1 illustrates the essential stages of the MLLC.

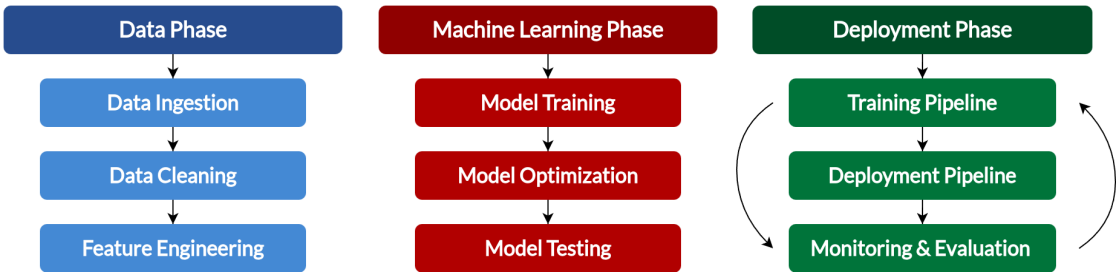


Figure 3.1: Machine Learning Life Cycle

3.1 Data Collection and Processing

Data serves as the foundational element of any Machine Learning (ML) pipeline, with its *quality*, *quantity*, *velocity*, *variety*, and *relevance* directly impacting model performance. This phase encompasses the collection of raw data from diverse sources, the validation of its integrity, and the transformation of this data into a structured and usable format. Proper data processing ensures that the resulting dataset is clean, consistent, and suitable for downstream tasks such as model training, validation, and evaluation. In the context of large-scale systems, the term Big Data refers to data that requires management in such volume, velocity, and variety that it exceeds the capabilities of traditional data processing techniques and tools. For instance, in industrial environments, data is often generated at high speed by sensors, machinery, or control systems. This continuous influx requires real-time or soft real-time processing capabilities, along with robust mechanisms to handle heterogeneous data types-structured, semi-structured, and unstructured. It is also important to recognize that such data may not always be accurate, complete, or reliable. Moreover, data characteristics and sources can evolve, necessitating adaptive strategies for data management and quality assurance.

To address these challenges, big data systems are commonly organized around a pipeline architecture, which consists of a series of layers-each responsible for a distinct stage in the data processing life cycle. The main stages typically include: *Ingestion*, capturing and importing data from multiple sources into the system; *Cleaning*, detecting and correcting errors, inconsistencies, and missing values to improve data quality. *Feature Engineering & Transformation*, deriving meaningful features and transforming raw data into formats suitable for ML models. Each of these stages plays a critical role in ensuring that data is processed efficiently and prepared effectively for analysis and decision-making.

Data Ingestion is the process of acquiring data from a wide variety of heterogeneous sources, such as databases, application programming interfaces (APIs), sensors, streaming platforms, and system log files. This step must account for diverse data formats, communication protocols, and input rates, which can range from periodic batches to high-throughput, real-time streams. To ensure robustness and reliability, the ingestion pipeline often incorporates mechanisms for fault tolerance, load balancing, and secure data transmission. Depending on the specific requirements of the application or system, ingestion can be executed in either batch mode, where data is collected and processed in fixed intervals, or streaming mode, where data is ingested and processed continuously as it arrives.

Data Cleaning is the phase following the ingestion phase. In this phase, data typically enters a cleaning phase designed to enhance its quality, consistency, and usability. This process involves several key tasks, including detecting and correcting errors, handling missing or inconsistent values, eliminating duplicate records, and standardizing data formats across sources. Effective data cleaning is critical for reducing noise and mitigating biases that could adversely affect the performance and reliability of downstream analytics or ML models. High-quality, well-prepared data ensures that models are trained on accurate and representative inputs, thereby improving their generalizability and trustworthiness in real-world applications.

Feature Engineering and transformation involve the selection, modification, and creation of new variables, known as features, from raw data to improve the performance and robustness of analytical and ML models. This phase is a critical bridge between raw data and model development, enabling algorithms to better capture underlying patterns and relationships. Effective feature engineering is often guided by domain expertise and requires a thorough understanding of both the dataset and the problem context. Common techniques include: Normalization and standardization to scale numerical features, En-

coding categorical variables (e.g., one-hot, label encoding), Time-series decomposition (e.g., trend, seasonality extraction), Dimensionality reduction (e.g., PCA, t-SNE), Extract statistical properties (e.g., computing mean, median, variance, skewness, and kurtosis etc.) Aggregation over time windows or categorical groups, Filtering irrelevant or outlier records, Joining datasets from multiple sources, Reshaping data structures (e.g., pivoting or unpivoting). Well-designed features can enhance predictive accuracy, model interpretability, and training efficiency. Conversely, poorly engineered features may introduce bias, noise, or lead to underfitting or overfitting. Following feature engineering, data transformation is performed to ensure compatibility with modeling frameworks and downstream applications. Together, feature engineering and transformation form an essential foundation for building accurate, reliable, and explainable ML models.

3.2 Model Selection

Model selection is a critical stage in the ML lifecycle, directly influencing the effectiveness and reliability of the resulting system. These processes involve identifying the most appropriate algorithm for a given task, configuring its hyperparameters, and optimizing it using available training data. The selection of a model depends on various factors, including: the data *Type & Modality* (e.g., tabular, time-series, image, or text), the *task want perform* (e.g., classification, regression, clustering), *performance requirements* such as accuracy, latency, and robustness, *Interpretability* needs in some applications, *computational constraints*, including processing power, memory, and training time. Once a suitable model is chosen, the training process involves adjusting the model's internal parameters to minimize a predefined loss function. This is typically achieved through iterative optimization techniques, such as gradient descent. The objective is to ensure the model captures meaningful patterns in the training data while maintaining the ability to generalize effectively to unseen instances, thus avoiding issues

like overfitting or underfitting.

3.2.1 Machine Learning

Machine Learning algorithms can be broadly classified according to how they interact with data and the level of supervision provided during training. The three primary paradigms are supervised learning, unsupervised learning, and reinforcement learning, each suited to different types of problems and data availability [19].

Supervised Learning involves training algorithms on labeled datasets, where each input is paired with a known output or target label. The model learns to approximate a mapping function from inputs to outputs by minimizing a predefined loss function, with the goal of generalizing well to unseen data. Typical supervised tasks include:

- *Classification*: Predicting discrete class labels (e.g., image classification).
- *Regression*: Predicting continuous numerical values (e.g., estimating values).

Common supervised algorithms include Decision Trees, Support Vector Machines (SVM), and Logistic Regression.

Unsupervised Learning is applied when the dataset lacks labeled outputs. The objective is to uncover hidden structures, patterns, or groupings within the data without prior knowledge of outcomes. Key applications include:

- *Clustering*: Grouping similar data points based on similarity or distance metrics, for example, customer segmentation using algorithms such as K-Means.
- *Dimensionality Reduction*: Reducing the number of input features while preserving essential information. This is particularly useful for visualization and mitigating the curse of dimensionality. Common techniques include PCA [20].

- *Anomaly Detection*: Identifying rare or abnormal instances that significantly deviate from expected patterns, such as in fraud detection or equipment fault diagnosis [21].

- *Feature Learning*: Automatically discovering informative representations or embeddings from raw data to enhance the effectiveness of subsequent ML tasks [22].

Reinforcement Learning is a learning paradigm in which an agent interacts with an environment and learns to make decisions by receiving feedback in the form of rewards or penalties [23]. Over time, the agent refines its behavior to maximize cumulative reward. RL is particularly effective in sequential decision-making problems, such as robotics, autonomous driving, and game playing. Core components of reinforcement learning systems include: *States*: Representations of the current situation; *Actions*: Possible decisions the agent can make; *Policy*: The strategy that maps states to actions; *Reward Function*: A mechanism for evaluating the desirability of an outcome.

3.2.2 Deep Learning

Deep Learning is a subfield of ML that uses multi-layered Artificial Neural Networks (ANN) to learn complex representations from data. These models consist of an input layer, multiple hidden layers, and an output layer. Each neuron receives weighted inputs from the previous layer, applies an activation function, and forwards the result. Deep architectures, such as Multilayer Perceptron (MLP), are capable of learning intricate patterns, especially from unstructured data like images, audio, or text.

The performance of a neural network depends heavily on the choice of activation functions, which introduce non-linearity and enable the model to approximate complex mappings. Common functions include sigmoid, tanh, ReLU, and softmax; their detailed mathematical definitions are provided in Appendix A. Training involves minimizing a loss function that quantifies the difference between predicted and true outputs. For re-

gression, losses such as MSE or MAE are typical, while classification tasks often use cross-entropy loss. (See Appendix B) for the corresponding formulas. Optimization is commonly performed using Gradient Descent, where model parameters are iteratively updated to reduce the loss. The training process includes initialization, forward pass, loss evaluation, backpropagation (based on the chain rule), and parameter updates. Advanced optimizers (e.g., Adam) are often used to accelerate convergence and stabilize learning.

Popular deep learning architectures include Convolutional Neural Networks (CNN) for spatial data, Recurrent Neural Networks (RNN) for sequences, Autoencoders for representation learning, and Transformers for context-aware sequential modeling. These architectures are applied across supervised, unsupervised, and reinforcement learning settings and are particularly suited for high-dimensional data and large-scale applications.

3.2.3 Generative AI

Generative AI [24] is a transformative area of ML centered on Deep Generative Models (DGM), which combines generative algorithms with deep learning to synthesize realistic, high-fidelity data that captures the distributional characteristics of training data. The primary goal of a DGM is to approximate an unknown probability distribution from limited i.i.d. samples, enabling both likelihood evaluation and the generation of new samples [25]. Several DGM architectures have emerged, including Generative Adversarial Networks (GAN), Variational Autoencoders (VAE), energy-based models, autoregressive models, and diffusion models.

Among classic oversampling techniques, SMOTE [26] generates synthetic minority samples using the K-NN algorithm, while ADASYN [27] emphasizes hard-to-learn regions of the minority class. Diffusion models [28], [29] operate via a two-step process: a forward pass progressively adds Gaussian noise to the input, and a reverse pro-

cess learns to denoise step-by-step, reconstructing realistic samples. TabDDPM [30] adapts this principle to model tabular data distributions, offering robust synthetic data generation for structured datasets. VAEs, particularly Tabular Variational Autoencoder (TVAE) [31], model the latent structure of tabular data, allowing synthetic sample generation through learned probabilistic mappings. TVAE retains statistical similarity to the real data while improving generative diversity. GANs encompass numerous variants [32], such as DCGANs and Conditional GANs. Wasserstein Generative Adversarial Network (WGAN) [33] introduces the Wasserstein distance as a loss function to address training instability and mode collapse, enhancing generator convergence and performance evaluation. Conditional Tabular GAN (CTGAN) [31] extends GANs to tabular data by integrating mode-specific normalization and enhancing categorical generation via a cross-entropy-driven loss. Its Training-by-Sampling mechanism ensures conditional diversity by matching real data distributions during generation.

3.2.4 Model Optimization

Model optimization is a crucial step in enhancing the performance and generalization ability of Machine Learning (ML) models. One of the most influential strategies in this phase is hyperparameter tuning, which focuses on adjusting algorithm-specific settings that govern the learning process. Hyperparameters are configuration variables set before training and are not learned from the data. They influence key aspects of model behavior, such as learning dynamics, model complexity, and regularization. In contrast to model parameters (e.g., weights in a neural network), hyperparameters must be manually specified or optimized through systematic experimentation. Common examples include the learning rate, batch size, number of hidden layers, regularization strength, and kernel type in support vector machines.

The goal of hyperparameter tuning is to identify the combination of hyperparameters that yields optimal model performance based on a chosen evaluation metric (see

Appendix D). Several popular hyperparameter tuning strategies include: *Grid Search* exhaustively explores all possible combinations within a predefined search space. While comprehensive, it is computationally expensive and impractical for high-dimensional parameter spaces. *Random Search* randomly samples hyperparameter combinations. It is generally more efficient than grid search, especially when only a few hyperparameters significantly impact performance. *Bayesian Optimization* utilizes a probabilistic model of the objective function (e.g., Gaussian Processes) and selects promising hyperparameters to evaluate based on an acquisition function. It is highly sample-efficient and suitable for exploring complex or expensive-to-evaluate search spaces. Effective tuning should be guided by validation performance and constrained by computational resources. To reduce the risk of overfitting to the validation set, it is advisable to employ techniques such as early stopping, regularization, and the use of a separate hold-out test set for final evaluation.

3.3 Model Training Strategies

Different approaches have emerged to address the trade-offs between computational efficiency, data availability, adaptability, and privacy. This section outlines three widely adopted paradigms, centralized learning, distributed learning, and continual learning, highlighting their characteristics.

Centralized Learning The most common paradigm is centralized learning, where all training data are aggregated into a single, central repository, where the model is trained using the complete dataset. This approach simplifies model management, allows for the direct application of global optimization techniques, and often achieves high predictive performance due to comprehensive data access. However, it may be impractical or undesirable when data are geographically dispersed, privacy-sensitive, or too large to transfer efficiently. Moreover, centralized architectures can become bottlenecks in large-scale

deployments, leading to high storage and communication costs as well as increased vulnerability to single points of failure.

Distributed Learning Distributed learning addresses the limitations of centralized approaches, namely privacy, scalability, and data locality, by training models across multiple computational nodes, each retaining its own subset of data. These limitations highlight the necessity of distributed learning techniques that enable model training across nodes while minimizing data movement and reducing reliance on a central server [34]. The most notable example is Federated Learning (FL) [35]–[37] which offers an alternative by enabling the training of ML models on large, distributed datasets without the need for data centralization. In this approach, a global model is disseminated to multiple participating nodes, where each node independently performs local training using its own data. Subsequently, the locally trained models or model updates are periodically aggregated by a centralized server, producing an updated global model. However, FL inherently relies on consistent and stable network connectivity between the central aggregation hub and participating nodes. Such environments can lead to synchronization issues due to varying update intervals, asynchronous node availability, and significant delays in model update aggregation. Furthermore, the effectiveness of FL is heavily influenced by the computational capacities, storage constraints, and hardware heterogeneity among edge devices in disrupted environments, which can slow training convergence, reduce model accuracy, and negatively affect overall system robustness [5], [38], [39].

Gossip-based learning [40] is a fully decentralized approach to distributed ML. In this paradigm, each participating node independently maintains a local model and intermittently communicates with a randomly selected subset of peers in the network. These communications involve the exchange of model parameters or updates, after which each node integrates the received information, often through weighted averaging or incremental updates, to refine its own model. Unlike federated learning, gossip learning

does not require a central coordinator, which enhances fault tolerance and allows it to function effectively in highly dynamic, unreliable, or partitioned environments. This makes it particularly suitable for settings such as disaster zones or tactical edge networks, where connectivity is intermittent and node availability may be unpredictable. However, gossip-based learning has its trade-offs. The peer-to-peer nature of communication, while enabling decentralization and robustness, can generate significant communication overhead, especially as the number of nodes grows. Frequent model exchanges between peers may lead to network congestion, particularly in bandwidth-constrained environments. Additionally, the lack of global synchronization can result in slower convergence and inconsistent model accuracy across nodes. Thus, while gossip learning offers strong resilience and adaptability in severe network conditions, its communication intensity and limited scalability for large or high-resolution models remain important challenges.

Google DeepMind introduced the Distributed Low Communication **(DiLoCo)** optimization algorithm, specifically designed for efficiently training large language models (LLMs) in distributed environments with limited connectivity [41]. DiLoCo is particularly suited for networks composed of isolated groups of devices, often referred to as "islands", where frequent, high-volume communication between nodes is impractical. Conceptually, DiLoCo builds upon the widely used federated averaging technique (FedAvg) but distinguishes itself by performing a significantly greater number of local training steps within each communication round. Internally, DiLoCo employs AdamW for local optimization steps (inner-loop optimization) and Nesterov momentum for aggregating and updating the global model parameters (outer-loop optimization). By maximizing local computation and reducing the frequency of communication rounds, DiLoCo mitigates the bandwidth and latency constraints inherent in poorly connected networks. Despite its advantages, the DiLoCo algorithm also has several notable limitations, acknowledged by the authors themselves. The primary assumption underlying DiLoCo is

that all participating devices (or workers) are computationally homogeneous, possessing similar hardware capabilities and computational performance. This inherent heterogeneity can cause synchronization bottlenecks when faster devices must wait for slower peers to complete their predefined number of local steps.

Continual Learning A very interesting approach, known as *Continual Learning (CL)* [42], [43], involves the development of ML models that can learn continuously from new data while retaining knowledge from previous tasks. This methodology, beyond its primary objective of enabling the model to learn continuously, also addresses the critical challenge of catastrophic forgetting. This phenomenon occurs when models lose previously acquired knowledge upon exposure to new data, effectively "forgetting" earlier information [44]. CL algorithms can be categorized into several techniques [45]: *Regularization-based* algorithms include methods like Elastic Weight Consolidation, where the primary idea is to prevent significant changes to important weights and parameters. *Adaptable and dynamic architectures and parameters* are another category, with examples such as Dynamic Filter Network, Layer Skipping, and Early Exiting. Additionally, there are *Replay-based* methods, including techniques like Experience Replay and Deep Generative Replay. So, unlike traditional ML models, which are typically trained once on a static dataset, CL models are dynamic and designed to learn incrementally from a continuous stream of incoming data. These models analyze data streams in real-time and update their parameters as needed to improve performance continuously. Continual Learning techniques have found application across a wide range of ML domains, including supervised learning for classification tasks [46], regression problems [47], object detection [48], [49], and also in the unsupervised learning domain [50]. This flexibility highlights the broad utility of continual learning in dynamic and data-rich environments.

3.4 Model Deployment

Model deployment is the process of integrating a trained ML model into a production environment, enabling it to generate predictions from real-time or batch data. This stage transforms the model from a static artifact into an operational component that supports decision-making, automation, and application integration. An effective deployment strategy must address scalability, latency, reliability, security, and maintainability to ensure alignment with the operational requirements of its deployed environment [51].

Cloud vs. Edge Deployment Two dominant architectural paradigms exist for model deployment: cloud-based and edge-based approaches. *Cloud deployment* centralizes computation within elastic, scalable infrastructure, making it ideal for high-throughput workloads and globally distributed applications. Cloud computing provides on-demand access to shared resources, such as servers, storage, and services, over standard network protocols. Its defining characteristics include broad network access, rapid elasticity for automatic scaling, and on-demand self-service for provisioning without human intervention. This model excels when centralized management, substantial computational capacity, and integration with cloud-native services are priorities. *Edge deployment* shifts computation closer to the data source, reducing latency, lowering bandwidth consumption, and enabling localized autonomy. By decentralizing processing to occur at or near where data is generated, edge computing minimizes dependency on continuous connectivity, improves responsiveness, and enhances privacy by limiting the transmission of sensitive information. In industrial contexts, this often involves embedding ML capabilities directly into machinery for on-site diagnostics, anomaly detection, and predictive maintenance. Typical hardware includes microcontrollers, embedded GPUs, and smart sensors capable of running lightweight inference models while interfacing with local or cloud-based orchestration systems.

In practice, the choice between cloud and edge deployment is rarely a matter of unrestricted preference. It is shaped by the available resources, operational constraints, and environmental conditions of the specific scenario, factors that inherently steer the decision toward the most viable solution, rather than leaving it entirely at the discretion of the system designer.

Model Performance Testing Performance testing is essential to verify that deployed ML models operate reliably and efficiently under real-world conditions. This process assesses both model-level performance, which focuses on predictive quality, and infrastructure-level performance, which addresses the efficiency of the serving environment.

At the model level, evaluation metrics such as accuracy, precision, recall, and F1-score measure the quality of predictions, while domain-specific indicators can capture task relevance in specialized applications.

At the infrastructure level, performance testing examines characteristics such as latency, throughput, scalability, and resource utilization. Latency measures the time between receiving an inference request and delivering a prediction, which can be critical in real-time or safety-critical applications. Throughput quantifies the number of inference requests the system can process within a given time frame, reflecting its ability to handle large-scale workloads. Resource utilization, including CPU, GPU, memory, and network usage, indicates the efficiency of the deployed environment and informs capacity planning.

Comprehensive performance testing combines these perspectives to provide an end-to-end view of system behavior, enabling teams to identify bottlenecks, optimize serving pipelines, and make informed decisions about model retraining, architecture adjustments, or infrastructure scaling. Effective monitoring and testing thus ensure not only the accuracy of the ML model but also the responsiveness and robustness of the entire deployment stack [52].

3.5 Multi-Criteria Decision Making

The primary goal of MCDM, a well-established and continuously evolving field of research, is to structure and simplify complex decision-making processes. It enables stakeholders to evaluate multiple, often conflicting, criteria and goals, incorporating subjective preferences that can influence the final solution. Over the years, numerous MCDM methods have been developed, refined, combined, and successfully applied across diverse domains, demonstrating their effectiveness. More recently, advanced data-driven techniques from artificial intelligence and fuzzy theory have been integrated to further enhance MCDM models [53]–[55].

MCDM methodologies are commonly categorized into *Multi-Objective Decision Making (MODM)* and *Multi-Attribute Decision Making (MADM)*, depending on the nature of the decision problem [53]. MODM problems involve an infinite set of possible solutions and multiple conflicting objectives that must be considered simultaneously, with the goal of identifying optimal trade-offs among these competing goals [56]. In contrast, MADM addresses problems with a discrete and finite number of alternatives, aiming to evaluate and select the best option based on multiple attributes.

In much of the literature, the term MCDM is used interchangeably with discrete MCDM (MADM). Such problems typically involve assigning weights to evaluation criteria and ranking alternatives according to these weighted scores. The following sections outline the typical steps in this process and highlight some of the techniques that can be applied. It is worth noting that many of these methods are, in principle, applicable to both MODM and MADM tasks, even though they may be more prevalent in one category.

3.5.1 Criteria Weighting

Determining the relative importance of each attribute is one of the most critical and challenging steps in MADM problems, as the assigned weights can heavily influence the final

decision [57]. The literature broadly categorizes weighting techniques into three classes: *Objective methods*: derive weights through mathematical or statistical analysis, without considering stakeholders' opinions. Conversely, *subjective methods* depend entirely on decision-makers' judgments and preferences. Finally, *integrated or combined weighting* combines data-driven analysis with stakeholder input to leverage the strengths and mitigate the weaknesses of both approaches.

Among the most widely adopted **objective algorithms**, the *Mean Weight method* [57] is notable for its simplicity, applying an equal-weighting scheme in which all criteria are assumed to be equally important. Another frequently used approach is the *Entropy Method* [57], which draws on probability theory to relate the importance of a criterion to the degree of uncertainty it encompasses. Criteria exhibiting broader value distributions entail greater uncertainty and are therefore considered more influential in the decision-making process. Similarly, the *Standard Deviation Method* [57] quantifies attribute importance by calculating their standard deviation, assigning higher weights to criteria with greater variability in their values. Overall, these methods provide a systematic, transparent, and replicable framework for weight determination, reducing potential biases inherent in subjective assessments. However, they also present certain limitations: their reliance on quantitative data may overlook contextual nuances and stakeholder preferences, and their computational demands can be significant for large or complex datasets. As a result, purely objective approaches may fail to fully capture the multidimensional nature of real-world decision problems.

In the domain of **subjective algorithms**, the *Point Allocation Method* is among the simplest. It provides decision-makers with a total of 100 points to distribute among the criteria according to their perceived priority, with higher point allocations indicating greater influence. A more sophisticated approach is the *Analytical Hierarchy Process (AHP)* [57], [58], which employs pairwise comparisons. In AHP, stakeholders evaluate each criterion against every other criterion using an ordinal scale, typically ranging from

1 to 9, to express relative importance. The resulting preferences are then processed to derive criteria weights, and a consistency ratio is calculated to verify the coherence of the judgments. A more recent and computationally efficient pairwise comparison method is the *Best Worst Method (BWM)* [59]. In BWM, the process begins by identifying the most and least influential criteria. Two rounds of comparisons follow: in the first, the best criterion is compared to all others; in the second, all criteria are compared to the worst one, both using an ordinal scale similar to that in AHP. By requiring fewer comparisons, BWM reduces computational effort and often yields more consistent judgments. Like AHP, it calculates a consistency ratio to assess the reliability of the resulting weights.

Subjective methodologies surpass objective ones in terms of flexibility and reduced computational demands, providing a straightforward means of incorporating personal preferences and priorities into the decision-making process. However, they are often less transparent and reproducible, and more vulnerable to biases and inconsistencies in judgment. As discussed above, each methodology has distinct advantages and limitations, and no single approach can be deemed universally superior. Consequently, different techniques are often combined to achieve more robust and optimized solutions. It is also worth noting that the methods presented here represent only a subset of the wide range of strategies developed over the years [57], [60].

3.5.2 Alternative Selection

The next stage involves evaluating the available alternatives using the previously assigned criteria weights and selecting the most appropriate option. Numerous methods have been proposed and refined for this purpose, some of which can also be applied to criteria weighting. Whether to address weight determination and alternative evaluation using the same method or two separate approaches depends on the problem's characteristics, the trade-offs involved, and the preferences of the decision-makers. In general, MADM techniques for alternative evaluation can be classified into three main categories:

pairwise comparison, outranking, and distance-based approaches [53], [61].

As introduced earlier, *pairwise comparison methods* include approaches such as AHP and its more general variant, the *Analytic Network Process (ANP)* [62]. These methods compare alternatives in pairs, incorporating the predetermined weights to assess their relative desirability. *Outranking methods*, which are particularly useful when dealing with incomplete or uncertain information, rank solutions by determining whether one alternative can be considered superior to another. Notable examples include *ELimination Et Choix Traduisant la REalité (ELECTRE)* and *Preference Ranking Organization METHod for Enrichment of Evaluations (PROMETHEE)*, both widely recognized and applied in practice. Finally, *distance-based methods* evaluate each alternative according to its proximity to the optimal solution, favoring those that are closest. Prominent techniques in this category include the *Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS)* and *VIseKriterijumska Optimizacija I Kompromisno Resenje (VIKOR)* [61].

3.5.3 Role in Machine Learning Life Cycle

MCDM methods play a pivotal role in optimizing various stages of the MLLC, especially in contexts where multiple, and often conflicting, alternatives must be evaluated simultaneously. By systematically weighing diverse criteria MCDM provides a structured framework for making informed, transparent, and justifiable decisions. Within the machine learning life cycle, MCDM techniques can be applied to several critical decisions. For instance, during model training, they can assist in selecting the most appropriate computational node by balancing factors such as processing capacity, memory availability, energy consumption, and cost. In model deployment, MCDM can guide the choice of hosting infrastructure, whether cloud, edge, or hybrid-by considering latency requirements, network reliability, security constraints, and budget limitations. Algorithm selection is another area where MCDM proves valuable, as it enables the compar-

ison of candidate models across multiple dimensions, including accuracy, interpretability, training time, and resource demands. Similarly, in feature selection, MCDM methods can prioritize subsets of features by accounting for their predictive power, relevance, redundancy, and computational impact. By applying MCDM across these stages, organizations can move beyond ad-hoc or single-metric decisions, instead adopting a holistic, criteria-driven approach. This is particularly valuable in environments where operational requirements, resource constraints, and performance targets must be simultaneously satisfied.

3.6 Challenges in Applying Machine Learning

Deploying Machine Learning presents several persistent challenges that hinder its seamless adoption. These challenges can be broadly grouped into three categories: data-related issues, ML model development and training processes, and infrastructure readiness. Addressing these challenges is essential for ensuring the successful, reliable, and sustainable integration of ML into industrial systems.

Data Challenges A key challenge in applying ML is the acquisition of high-quality, representative datasets. Although these environments are typically equipped with dense sensor networks and advanced monitoring systems, the collected data may still be affected by inconsistencies, noise, missing values, or significant class imbalance, particularly in the case of rare fault events. In many applications, historical datasets are either limited in scope or lack systematic labeling, constraining the applicability of supervised learning methods and reducing model reliability. In addition to technical quality issues, concerns related to data privacy, ownership, and governance often emerge, especially when information must be shared across organizational units or with external vendors. Maximizing the value of data-driven innovation in such contexts requires not only sophisticated analytics pipelines, but also well-defined data governance frameworks, se-

cure data-sharing protocols, and robust pre-processing and augmentation strategies to ensure completeness, accuracy, and representativeness.

Model Challenges The integration of Machine Learning introduces substantial complexity across model development, deployment, and lifecycle management. Managing the complete ML pipeline, including training, evaluation, version control, storage, deployment, and updates, can be resource-intensive, demanding both specialized expertise and robust pipeline architectures, including software, computational resources, and algorithmic frameworks. Once operational, models must be continuously monitored to ensure accuracy, stability, and relevance, particularly under evolving production conditions, equipment modifications, or shifts in input data distributions.

Infrastructure Challenges: The underlying infrastructure plays a decisive role in the performance and reliability of Machine Learning applications. A key consideration is the choice of deployment location, cloud, edge, or a hybrid approach, each with specific trade-offs in latency, computational capacity, energy consumption, and network dependence. Edge deployment enables fast, low-latency inference and greater operational autonomy but is limited by processing power, memory, and storage. Cloud deployment offers scalability and centralized management for computationally demanding tasks but relies on stable connectivity and may introduce latency that is unsuitable for time-critical processes. Regardless of the chosen architecture, an effective platform for orchestration and monitoring frameworks is essential.

3.7 Chapter Summary

This chapter has outlined the complete machine learning life cycle, illustrating an overview of data collection and processing, emphasizing the critical importance of high-quality, representative datasets. Key stages such as data ingestion, cleaning, and feature engineering were discussed as foundational steps for enabling effective model training. The model selection section explored different algorithmic approaches, including traditional Machine Learning, Deep Learning, and Generative AI, alongside strategies for model optimization. Special attention was given to aligning model capabilities with operational requirements, data characteristics, and computational resources. Model training strategies were examined with a focus on distributed learning and continual learning, both essential for scaling solutions and adapting to changing conditions in dynamic production environments. This led into the model deployment discussion, where cloud-based and edge-based, were compared, and the importance of model performance testing was highlighted. The chapter also introduced Multi-Criteria Decision Making as a systematic approach to guide complex choices within the ML life cycle. MCDM methods were presented as valuable tools for weighting decision criteria going beyond ad-hoc or single-metric decisions, instead adopting a holistic, criteria-driven approach. These components provide a structured background for designing, implementing, and managing Machine Learning systems from data acquisition to operational deployment.

Subsequent chapters will more precisely assess the risks, opportunities, and design patterns associated with deploying machine learning under conditions where failures are unacceptable and responsiveness is crucial.

Table 3.1: Summary of Machine Learning Life Cycle phases, objectives, techniques, and challenges

MLLC Phase	**Objective**	**Representative Techniques**
Data Processing	Acquire, clean, and prepare data for model development	Data acquisition systems, preprocessing, normalization, feature extraction
Model Selection	Identify appropriate learning algorithms and architectures	Classical Machine Learning, Deep Learning, Generative AI
Model Training Strategies	Train models efficiently under operational constraints	Centralized training, distributed learning, continual learning, transfer learning
Model Deployment	Deploy models for real-time or near-real-time operation	Cloud deployment, edge deployment, hybrid cloud–edge architectures
Lifecycle Management	Ensure long-term reliability, adaptability, and robustness	Monitoring, retraining, validation, lifecycle management pipelines

Chapter 4

Machine Learning in High-Stakes Environments

Machine Learning has emerged as a transformative technology in domains characterized by complexity, uncertainty, and high operational stakes. Its ability to extract patterns from large and heterogeneous datasets, adapt to dynamic conditions, and enable predictive as well as prescriptive decision-making has positioned it as a critical enabler of next-generation high-stakes systems. However, the adoption of Machine Learning in these environments is not without challenges, and strict requirements must be carefully tailored to the unique operational constraints identified in the previous chapter.

4.1 ML Applications in High-Stakes Environments

The application of Machine Learning has extended across all stages of the production life cycle, from predictive maintenance and quality assurance to workflow optimization and adaptive process control [2], [8].

Predictive maintenance leverages sensor data, such as vibration, temperature, acoustic, and pressure measurements, to detect early indicators of wear, misalignment, or component failure. The availability of high-frequency, high-fidelity data streams enables near real-time analysis and frequent model retraining, often facilitated by cloud-based pipelines. Such models estimate asset health, predict remaining useful life (RUL), and reduce unplanned downtime by enabling timely interventions. A central component of predictive maintenance is anomaly detection [21], which identifies deviations from expected equipment behavior that may signal early-stage degradation or impending faults. These anomalies, detected through statistical modeling or ML-based pattern recognition, allow maintenance activities to be scheduled proactively, minimizing operational disruptions and associated costs.

Machine Learning can also enhance quality control by automating defect detection and process monitoring through statistical modeling, computer vision, and sensor-based analysis. Integrated directly into production machinery, these systems enable real-time quality assurance, flagging faults and dynamically adjusting process parameters to maintain product consistency. Quality control applications focus on verifying that products meet predefined standards by analyzing visual data, process parameters, and sensor signals. Beyond defect detection, ML supports process optimization by identifying parameter ranges that maximize efficiency and product quality while minimizing costs. Through continuous feedback and adaptive control, these systems help sustain high output standards and operational efficiency under varying production conditions.

In disaster recovery and tactical environments, Machine Learning plays a pivotal role in enhancing situational awareness, decision support, and operational coordination under highly dynamic and resource-constrained conditions [5]. A key application area is rapid damage assessment, where computer vision models process satellite and aerial imagery to identify collapsed structures, flooded areas, or blocked transportation routes [11]. Such models enable emergency responders to prioritize high-risk zones and allo-

cate resources more effectively. Another critical use case is search-and-rescue, where Machine Learning-enabled unmanned aerial vehicless and autonomous ground vehicles analyze sensor feeds to detect survivors, hazardous materials, or structural instabilities in real time. These systems rely heavily on anomaly detection, pattern recognition, and multimodal data fusion, integrating inputs from thermal imaging, LiDAR, acoustic sensors, and social media streams, to improve coverage in environments where communication networks may be degraded. Furthermore, Machine Learning supports tactical decision-making within Command and Control (C2) frameworks by processing heterogeneous data streams, estimating uncertainties, and recommending courses of action under time pressure.

4.2 Implications on Machine Learning Systems

Deploying Machine Learning systems in High-Stakes Environments faces strict constraints that affect every stage of the system life cycle, from design and training to deployment and long-term maintenance. Often requiring that algorithmic design choices are closely aligned with domain-specific limitations and constraints, ensuring both legal compliance and operational reliability. These considerations highlight two main areas where constraints are most pronounced: **(i)** the training of models under limited resources, scarce data, and domain-specific restrictions, and **(ii)** the deployment of those models in unpredictable or resource-constrained environments. Addressing both areas is essential to ensure that ML systems remain effective, resilient, and safe throughout their operational lifetime, and enabling adaptive mechanisms that can respond to evolving regulatory, environmental, or operational conditions.

4.2.1 Training Constraints Across Operational Contexts

The training of Machine Learning models in High-Stakes Environments is fundamentally shaped by infrastructural conditions, most notably network reliability and bandwidth availability. These factors directly determine how data is collected, transmitted, and processed during the training life cycle, influencing both the architecture of training pipelines and the strategies employed to adapt models over time.

Network reliability dictates the stability with which training data, model checkpoints, and configuration updates can be exchanged between distributed nodes or transferred from edge devices to centralized infrastructure. In highly reliable networks, large volumes of data can be streamed continuously, enabling real-time synchronization of training progress, seamless aggregation of updates, and rapid model iteration. By contrast, environments with intermittent or unstable connectivity suffer from frequent disconnections that disrupt training workflows and render centralized approaches impractical.

Bandwidth limitations compound these challenges by constraining both the volume and speed of data transmission. Low-bandwidth links delay the upload of raw data, the retrieval of pre-trained models, and the exchange of updates in collaborative training. Such bottlenecks increase the time needed for the training process and often necessitate compromises in dataset size, model complexity and size, or update frequency. To counteract these effects, techniques such as data compression, selective sampling, and adaptive synchronization become essential, allowing systems to operate effectively despite limited throughput.

Generally, different training paradigms emerge in response to these constraints. In well-connected environments, centralized cloud-based training remains the most efficient solution, leveraging high-performance infrastructure for rapid retraining, large-scale model development, and centralized oversight. In partially connected and bandwidth-limited settings, hybrid strategies are often adopted, where data is collected and prepro-

cessed locally, transferred to the cloud in batches, and integrated through asynchronous checkpointing and partial synchronization. These approaches strike a balance between centralized efficiency and tolerance of network variability. In severely constrained contexts, training must be performed entirely at the edge or on local devices. Methods such as Federated Learning enable models to adapt autonomously without relying on persistent connectivity or large-scale data transfers; however, in environments with minimal computational resources, energy restrictions, or highly unstable communication links, even Federated Learning may become impractical, therefore needing ad-hoc strategies.

Ultimately, the effectiveness of training in High-Stakes Environments depends on aligning the chosen paradigm with the operational realities of deployment. Achieving resilient and adaptive Machine Learning under these conditions requires not only robust algorithmic techniques but also a deep knowledge of the infrastructural and logistical boundaries that define what is feasible in practice.

4.2.2 Deployment Constraints Across the Operational Contexts

The deployment of Machine Learning models in High-Stakes Environments is shaped by the dynamics of network latency, reliability, and bandwidth availability. These factors determine where inference can occur, how updates are propagated, and the degree to which computation can be distributed across edge and cloud environments. Unlike traditional settings, deployment in such environments is not a straightforward one-time event but an ad-hoc, continuous process, requiring dynamic updates, rollback capabilities, and configuration management, all while avoiding any disruption to critical operations.

Network latency, particularly when moderate to high, often requires that inference and decision-making be carried out near the data source. This is typically achieved by deploying models on edge devices or local servers, which helps minimize delays caused by round-trip communication with cloud infrastructure. Conversely, when latency constraints are minimal, cloud-based inference becomes a practical alternative, offering ad-

vantages in scalability, centralized monitoring, and streamlined maintenance workflows.

Network reliability plays a decisive role in shaping deployment strategies. In highly reliable networks, continuous integration and frequent model updates are achievable. In contrast, intermittent or unstable connectivity demands workflows that support asynchronous delivery, tolerate interruptions, and rely on robust fallback strategies. Bundling model updates, containerized packaging, and edge-based inference pipelines represent practical solutions under such conditions.

Bandwidth limitations, although frequently regarded as secondary to reliability and latency, nonetheless exerts a substantial influence on the feasibility of deployment. Limited bandwidth constrains the frequency and size of updates, making techniques such as model compression, quantization, and differential updates essential for reducing transmission overhead. These limitations directly inform model architecture decisions, often requiring trade-offs between performance, robustness, and resource efficiency.

Different deployment patterns emerge depending on the operational context. In well-connected environments with stable infrastructure, cloud deployment supports large-scale models, centralized maintenance, and frequent updates. In bandwidth- or reliability-constrained settings, hybrid approaches are often favored: training and coordination remain in the cloud, while inference is executed at the edge to meet latency demands. In the most restricted scenarios, fully local deployment is necessary, with models designed to be self-contained and capable of operating autonomously for extended periods. Updates in these contexts, if possible at all, are opportunistic and must be executed with minimal reliance on persistent connectivity.

Ultimately, effective deployment in High-Stakes Environments requires aligning system capabilities with domain realities. The choice among cloud, edge, or hybrid approaches depends not only on infrastructure but also on mission-critical demands for responsiveness and resilience, necessitating strategies that balance technical feasibility with contextual risks and responsibilities.

Table 4.1: Impact of the Connectivity Axes on Training and Deployment. Network reliability refers to connection uptime and packet delivery stability; bandwidth reflects sustainable throughput for ML-related data transfers; latency captures time-to-response when accessing remote resources.

Connectivity Axis	**Impact on**	**Explanation**
Network Reliability	Training & Deployment	Unstable or intermittent connections hinder data transfer during training, delay or prevent frequent model updates, and disrupt real-time inference from cloud-based models. High reliability enables centralized workflows. Low reliability necessitates local autonomy and asynchronous update strategies.
Bandwidth Availability	Training & Deployment	Limited bandwidth constrains uploading large training datasets and downloading updated models or parameters. In deployment, it restricts the frequency and size of updates, often requiring model compression, quantization, or transmission of only aggregated data.
Network Latency	Deployment	High latency to remote resources impacts responsiveness, making cloud inference impractical for real-time systems. Low latency enables centralized inference, while high-latency contexts require edge on-device deployment to ensure timely decision-making.

4.3 A Typology Framework for ML in High-Stakes

Based on the numerous use cases examined throughout the doctoral research, including collaborations with more than five world-leading companies in the development of mission-specific Machine Learning applications, it has become evident that ML usage in High-Stakes Environments does not conform to a single architectural or operational paradigm. Instead, deployments span a wide spectrum of infrastructural conditions, organizational constraints, and mission-critical requirements. To support both analytical reflection and system design, it is therefore necessary to establish a coherent framework that captures this diversity while providing a structured basis for comparison across different use cases.

A useful starting point is the classification of operational contexts along three key dimensions: network reliability, latency and bandwidth availability. Mapping these technical axes to their direct impact on Machine Learning workflows allows system designers to reason explicitly about trade-offs, balancing performance, resilience, and resource efficiency. The typology developed here is grounded in empirical observations, technical evaluations, and iterative modeling conducted across diverse operational domains throughout this doctoral research. This framework does not aim to exhaustively capture every conceivable deployment scenario. Instead, it abstracts recurring patterns that emerge when infrastructural conditions intersect with mission requirements. Each category within the typology is defined by the interplay of three core factors: the reliability of communication infrastructure, the availability of bandwidth, and the degree to which inference tasks are latency-sensitive. In combination, these factors shape critical design decisions, such as where models are trained, where inference is executed, how frequently updates can be delivered, and how robust systems must be against infrastructural degradation or outright failure. Rather than examining these factors in isolation, the framework integrates them into scenario types that mirror real-world deployments.

From a technical perspective, this synthesis yields three recurrent training-deployment profiles that span the majority of high-stakes operational contexts.

Cloud Training & Cloud Deployment

In this configuration, both training and inference occur entirely in the cloud. It is most suitable for applications with highly reliable networks, ample bandwidth, and relaxed latency requirements. Cloud-centric workflows enable centralized data aggregation, large-scale model training, and seamless deployment without local computational constraints. A representative example is predictive maintenance in smart manufacturing plants, where connectivity is stable and near-real-time responses are sufficient.

Cloud Training & Edge Deployment

Here, model training is performed in the cloud to leverage large-scale computational resources, but inference is executed locally on edge devices. This profile is preferred when low-latency responses are essential but reliable network connectivity is still available for periodic updates. Models must often be compressed, quantized, or pruned to fit edge hardware efficiently. Industrial machinery on the shop floor, equipped with embedded ML models that adapt operations in real time while receiving periodic updates from the cloud, exemplifies this hybrid approach.

Edge Training & Edge Deployment

In the most autonomous configuration, both training and inference are carried out entirely at the edge. This setup becomes essential in environments with unreliable connectivity, constrained bandwidth, or high-latency networks, where dependence on centralized resources is impractical. Edge-native training enables rapid adaptation to local data conditions without requiring continuous cloud access. For instance, onboard vision systems for autonomous UAVs operating in disaster zones must function with intermittent

or nonexistent connectivity, making local training and inference indispensable.

4.3.1 High-Stakes Environments Types

The three technical profiles introduced above can be situated within a three-dimensional space defined by network reliability, bandwidth availability, and latency to remote servers. **Type 1** corresponds to the high-reliability, high-bandwidth, and low-to-moderate-latency region. Under these conditions, large-scale data transfers and centralized inference are feasible without compromising responsiveness. **Type 2** lies in the middle range across all three axes. Connectivity is generally stable but not fully guaranteed, bandwidth is sufficient though occasionally restrictive, and latency requirements are tighter than what pure cloud deployment can consistently satisfy. This hybrid setup can exploit the scalability of centralized training while relying on local inference for responsiveness, often requiring model compression and optimization to fit edge hardware. **Type 3** occupies the low-reliability, low-bandwidth, and high-latency to the remote cloud region. In such settings, frequent cloud interaction is infeasible, making edge-based training and inference the only viable option. These conditions typically arise in remote or resource-constrained environments where systems must operate with autonomy and minimal reliance on external infrastructure. Table 4.2 summarizes the distinguishing network characteristics across the three types.

Table 4.2: Network Characteristics by High-Stakes Environments Type

HSE Types	**Reliability**	**Latency to Remote**	**Bandwidth**
Type 1	High	Low-Moderate	High
Type 2	Moderate	Moderate-High	Moderate
Type 3	Low	High	Low

While these profiles capture the technical dimensions of training and deployment, their significance becomes clearer when grounded in real-world high-stakes domains.

Importantly, the archetypes are neither mutually exclusive nor static: systems may shift across categories over time or combine features of multiple types within a broader architecture. Beyond purely technical factors, external influences such as regulation, organizational priorities, and environmental conditions also determine how a system is deployed. In addition, sudden disruptions, whether from adversarial attacks, infrastructure failures, or urgent mission demands, can rapidly shift a system from one type to another, necessitating adaptation over time. Type 1 is exemplified in smart manufacturing environments, where reliable connectivity and abundant resources act as ML enablers. Type 2 arises in in-field industrial environments, where partial connectivity and variable resources impose ML constraints that require careful balancing between cloud and edge. Type 3 emerges in disaster-impacted environments, where autonomy under degraded or contested infrastructure highlights the need for adaptive ML. Together, these domains provide complementary perspectives on how the typology informs the design of context-aware, technically feasible, and operationally robust deployment strategies.

4.4 Type 1: Smart Manufacturing Environments

As a cornerstone of the Industry 4.0 and 5.0 paradigms, smart manufacturing integrates data-driven insights with sustainability, resilience, and human-centered design [1]–[3]. Beyond productivity and automation, strategic objectives include zero-waste and zero-defect production [63], aiming to reduce material use, minimize energy consumption, and lower environmental impact while maintaining uncompromised quality and consistency. Zero Defect Manufacturing (ZDM) advances quality management by moving from reactive quality control toward predictive and preventive strategies that target "first-time-right" outcomes. This transition not only enhances competitiveness but also positions industry practices within broader sustainability agendas. In this context, machine learning is not merely an accelerant but a foundational enabler. Applications

such as predictive maintenance, supply chain optimization, and adaptive process control demonstrate how ML drives operational efficiency while embedding human welfare and environmental responsibility as central design principles.

4.4.1 ML Enablers in Smart Manufacturing Environments

Smart manufacturing environments exhibit four interrelated characteristics that collectively enable the adoption of Machine Learning models:

Stable Connectivity

Continuous and reliable communication between edge devices, sensors, and centralized platforms and servers enables scalable data acquisition, real-time analytics, and seamless process coordination.

High Computational Resources

These facilities are typically equipped with on-premise servers or cloud-based infrastructures offering substantial computational capacity. Such resources support the training of complex ML models, large-scale data processing, and the deployment of both lightweight and large Machine Learning algorithms.

Controlled Operating Conditions

Unlike dynamic or unpredictable field settings, smart manufacturing operates under tightly regulated conditions. This predictability reduces variability and noise in the collected data, making it an ideal environment for deploying ML algorithms and optimization techniques with high stability and reliability.

Closed-Loop Feedback

Integrated feedback mechanisms automatically adjust process parameters in real time based on sensor data and ML inferences. This closed-loop control facilitates continuous process optimization, rapid anomaly mitigation, and consistent quality assurance with minimal human intervention in tasks that are repetitive or require levels of precision best achieved by machines.

These characteristics make type 1 environments particularly well-suited for the deployment of advanced Machine Learning models across a diverse range of applications. The synergy of stable infrastructure, reliable data acquisition, and abundant computational resources facilitates the implementation of both diagnostic and predictive analytics. Such capabilities enable continuous process improvement, enhance operational efficiency, and support informed decision-making throughout the entire production life cycle.

4.5 Type 2: In-Field Industrial Environments

Industrial machine environments also include oil rigs, wind farms, mining sites, and specialized workshops. Unlike smart factories that operate under tightly stable controlled indoor conditions, these sites are often geographically dispersed and located closer to end users. Equipment is typically deployed in outdoor or semi-remote settings yet remains connected, most often through relatively reliable cellular networks, to a centralized, cloud-based platform.

4.5.1 ML Constraints in In-Field Environments

Even in these distributed settings, Industry 5.0 principles such as zero-defect and zero-waste manufacturing remain relevant, though typically pursued at the scale of individual

machines rather than entire factories. To achieve this, machines are engineered for autonomous and semi-autonomous operation, ensuring reliable performance even when internet or cloud connectivity is intermittent. On-device processing ensures faster response times, resilience to short-term disruptions, and stable day-to-day functionality. Despite these strengths, three interrelated constraints shape ML deployment in in-field environments

Non-Guaranteed Stable Connectivity

Connectivity in these environments is generally robust but vulnerable to disruptions from environmental conditions, equipment interference, or network congestion. Interruptions are often brief yet must be anticipated in system design, particularly for safety-critical operations.

Moderate Bandwidth with Occasional Constraints

Available bandwidth is sufficient for continuous monitoring and low-volume data exchange but can become restrictive during large-scale transfers such as high-frequency sampling or bulk dataset uploads. These limitations necessitate careful scheduling of high-volume transmissions to avoid degrading operational traffic or exceeding bandwidth usage thresholds.

Edge Hardware Limitations

Edge devices deployed in these settings often operate under constraints in processing power, memory, and energy availability. As a result, computationally intensive tasks such as model training are performed in the cloud, with trained models deployed to the edge for local inference. To meet performance requirements under hardware constraints, models are optimized using techniques such as quantization, pruning, and efficient architecture design, balancing accuracy with responsiveness in resource-limited conditions.

Taken together, these constraints define Type 2 environments as hybrid settings. They combine the scalability of centralized training with the responsiveness of local inference, but demand deliberate trade-offs in model design, deployment, and data management to remain effective under such constraints.

4.6 Type 3: Disaster-Impacted Environments

Disaster-impacted environments are defined by very limited power and communication infrastructure. Responders must often rely on portable or ad-hoc resources, such as rugged laptops, drone-mounted edge nodes, or micro-servers, to sustain even minimal digital capability. Unlike types 1 and 2, which can exploit stable or intermittent connectivity, Type 3 assumes prolonged or total isolation from centralized infrastructure, with only opportunistic or delay-tolerant networking available. These scenarios arise in natural and human-induced crises, including wildfires, floods, earthquakes, armed conflict, and large-scale cyber attacks, as well as during remote humanitarian operations. What distinguishes them from other types is not simply degraded performance but the near-complete collapse of infrastructure, rendering traditional network independent strategies infeasible. The defining feature is the prevalence of Disrupted, Degraded, Intermittent, and Low-Bandwidth conditions, ranging from temporary slowdowns to complete blackouts, under which systems must remain autonomous and resilient [64].

Operating in these environments requires more than robust engineering; it demands the adaptation of Machine Learning workflows to function under unpredictable constraints and without reliable external support [65]. Research in this domain highlights the urgency of autonomy, with priorities centered on minimizing fatalities, enabling rapid rescue, delivering medical assistance, and ensuring safe evacuations [10], [11], [66]. Ultimately, Type 3 environments underscore the critical role of adaptive ML in sustaining lifesaving operations when conventional infrastructures fail.

4.6.1 ML Adaptability in DDIL Environments

Operating in DDIL conditions imposes constraints far more severe than those encountered in industrial or in-field environments. In these contexts, intelligent systems must contend with restricted access to critical data due to broken backhaul links and disrupted pipelines, unreliable or damaged technologies such as malfunctioning sensors and disabled base stations, and fragile hardware with limited power availability. Security risks are also heightened, as unstable or improvised communication channels become vulnerable to interception or cyber exploitation. These characteristics mean that systems cannot depend on centralized cloud resources, nor can they assume stable connectivity or sufficient bandwidth. Instead, power is often scarce, infrastructure is degraded, and the environment itself is volatile. To ensure operational continuity, Machine Learning must not only run at the edge but also adapt to survive under isolation, scarcity, and unpredictability [5].

The fusion of large-scale data analytics with ML technologies remains crucial even in these settings, as it enables the extraction of actionable insights that can guide decision-making and improve disaster management outcomes. However, the challenges of real-time data acquisition and model training in resource-constrained conditions make the deployment of ML far from straightforward. At the same time, types 1 and 2 of heterogeneous environments can rely on inference at the edge with periodic synchronization to the cloud, type 3 demands fundamentally new strategies. In disaster-impacted environments, edge intelligence must extend beyond inference to encompass local and distributed training, often conducted opportunistically and with strong adaptability to intermittent resources. This shift is not merely a matter of coping with constraints but of ensuring the very survival and continuity of intelligent systems under the most extreme operational conditions. The following subsections examine these challenges in detail and discuss the adaptations required for ML to function effectively in DDIL environments.

Mitigating Connectivity Collapse

In disaster-impacted environments, communication infrastructure is frequently damaged or destroyed, leading to extended periods of isolation. Unlike in less severe contexts, where intermittent links still permit synchronization with remote servers, DDIL conditions often involve complete communication blackouts that can last from some minutes to several hours. Such outages prevent centralized training updates, remote monitoring, and reliance on cloud-based inference, forcing systems to operate autonomously for prolonged periods. To cope with these disruptions, reliance shifts toward opportunistic and ad-hoc networking. Technologies such as ad-hoc mesh networks and Mobile Ad-hoc Networks (MANETs) enable direct device-to-device communication, bypassing centralized infrastructure. While these networks are inherently unstable and highly dynamic, they provide critical pathways for short-range coordination and intermittent synchronization when opportunities arise.

Under these conditions, only the most essential information should be shared to minimize overhead, while preserving autonomy during extended periods of disconnection. Machine Learning systems designed for DDIL environments must therefore integrate delay-tolerant mechanisms that allow data and model updates to be cached locally and exchanged asynchronously when connectivity is briefly restored. Adaptations include local decision-making, cached or rolling model updates, and asynchronous synchronization strategies that reconnect opportunistically without interrupting ongoing processes. By reducing dependence on centralized infrastructure, such systems can maintain functionality and continuity even in the face of complete connectivity collapse.

Managing Extreme Bandwidth Limits

When connectivity is available in disaster-impacted environments, it is typically severely constrained, marked by low throughput, high latency, and frequent instability. Communication often occurs in brief bursts, facilitated by satellite relays, drone-mounted

base stations, or improvised mesh and radio networks. Such channels are fundamentally unsuited for large-scale data transfers, continuous sensor streaming, or centralized model retraining, directly limiting coordination and situational awareness across response teams. In these conditions, communication prioritization becomes essential. Safety-critical alerts, anomaly detections, and operational control messages must take precedence over non-urgent transmissions such as routine monitoring logs or bulk dataset uploads. This ensures that scarce capacity is preserved for mission-critical functions, where even small delays can endanger human safety.

To remain effective under such restrictions, Machine Learning systems must adopt communication-efficient strategies. Instead of transmitting raw sensor data or complete model states, systems should send compressed, lightweight updates limited to the most informative elements, such as model deltas, anomaly flags, or aggregated summaries. These selective exchanges provide actionable insights without overwhelming limited bandwidth. Non-essential data can be cached locally, enabling deferred synchronization once more stable or higher-capacity connections become available. By restructuring communication into prioritized, opportunistic, and bandwidth-aware exchanges, ML-driven systems can maintain reliability and relevance even under extreme bandwidth constraints.

Engineering for Hardware and Power Fragility

Disaster-impacted environments rarely provide stable power or robust infrastructure, forcing operations to rely on portable, ruggedized, and energy-limited devices. These include hardened laptops, drone-mounted processors, low-power embedded systems, or field-deployable micro-servers. Under such constraints, energy-aware models become indispensable [67]. Models must be designed to minimize computational overhead and memory usage while preserving performance, ensuring that inference and even limited on-device training can be executed without exhausting battery reserves or overloading lo-

cal hardware. Techniques such as model compression, quantization, pruning, and adaptive inference pipelines are essential for maintaining efficiency under strict power budgets.

4.7 Guidelines for Classifying High-Stakes Environments

To apply the proposed typology in practice, practitioners require a systematic way of assessing which environment type best matches their operational context. While the three dimensions-network reliability, bandwidth availability, and latency sensitivity-form the technical foundation, a structured assessment should also include guiding questions and requirement checks.

Key Questions to Ask

- Network Reliability: How stable is connectivity between edge devices and remote servers? Are interruptions rare, occasional, or frequent?
- Bandwidth Availability: Is sufficient bandwidth consistently available to support high-volume data transfers, or is it constrained to monitoring and control traffic?
- Latency Sensitivity: Are inference tasks safety- or time-critical, requiring near-real-time responsiveness, or can they tolerate delay?
- Infrastructure Dependencies: To what extent does the system depend on centralized cloud infrastructure, and what contingencies exist if that connection is lost?
- Environmental Stability: Are operating conditions predictable (e.g., controlled environments), partially variable (e.g., outdoor industrial sites), or highly unstable (e.g., disaster zones)?

Requirements to Verify

- Type 1 (Enablers): Confirm the presence of high-reliability, high-bandwidth connections, abundant compute capacity, and stable operating conditions.
- Type 2 (Constraints): Verify that connectivity is available but not guaranteed, bandwidth is sufficient for routine operations but limited for bulk transfers, and edge devices impose compute or memory constraints.
- Type 3 (Adaptability): Assess whether communication is degraded, intermittent, or absent; confirm that systems must operate autonomously under Disrupted, Degraded, Intermittent, and Low-Bandwidth conditions; and ensure ML workflows are adapted for resilience and lightweight execution.

By answering the questions presented in 4.3 and verifying the corresponding requirements, system designers can classify their environment into one of the three types, or a hybrid form that combines aspects of multiple categories. This structured assessment provides a practical entry point for reasoning about where training should occur, how inference should be deployed, and what degree of adaptability is required for robust performance.

Table 4.3: Sample Checklist for Classifying High-Stakes Environments

Guiding Question	**Type 1 (Enablers)**	**Type 2 (Constraints)**	**Type 3 (Adaptability)**
Is connectivity stable and continuous?	Always stable	Generally stable but occasional interruptions	Frequently unstable or absent
Is bandwidth sufficient for large transfers?	High, supports bulk transfers seamlessly	Moderate, sufficient for monitoring but limited for bulk uploads	Low, limited to minimal or opportunistic data exchange
How latency-sensitive are inference tasks?	Low to moderate latency tolerated without impact	Moderate to high sensitivity; requires partial local inference	High sensitivity; only local inference feasible
What is the compute capacity at the edge?	Abundant local/cloud compute resources available	Limited edge compute; cloud training with edge inference	Highly constrained devices; lightweight edge-only models
What are the operating conditions?	Controlled, predictable factory-like settings	Semi-controlled, outdoor/industrial environments	Unstable, disaster-impacted or austere environments

4.8 Chapter Summary

This chapter examined the role of Machine Learning in high-stakes environments, highlighting both its opportunities and inherent limitations. We began by outlining representative applications and their implications for system design, focusing on how training and deployment are constrained by operational contexts. This provided the foundation for a typology framework that classifies environments along three key dimensions: network reliability, bandwidth availability, and latency sensitivity. Within this framework, three archetypal settings were introduced to illustrate recurring patterns of deployment. Type 1 environments, exemplified by smart manufacturing, emphasize the presence of strong enablers such as stable connectivity, abundant computational resources, and closed-loop feedback systems. Type 2 environments, represented by in-field industrial settings, highlight the constraints that emerge from non-guaranteed connectivity, moderate bandwidth, and edge hardware limitations, requiring deliberate trade-offs in model design and deployment. Type 3 environments, characteristic of disaster-impacted and DDIL contexts, underscore the need for adaptability, as systems must sustain autonomy and resilience in the near-total absence of infrastructure.

The framework is grounded in real-world use cases. The following three chapters each expand on one archetype through a representative deployment domain: Chapter 5 focuses on smart manufacturing environments and the enablers that allow ML at scale; Chapter 6 examines in-field industrial environments, emphasizing the constraints that shape hybrid deployment strategies; and Chapter 7 explores disaster-impacted environments, highlighting how I adapted ML approaches under degraded or denied infrastructure. Collectively, these chapters demonstrate how the typology translates into practice, providing concrete illustrations of how ML can be designed, deployed, and sustained in high-stakes operational domains.

Chapter 5

Type 1: Smart Manufacturing Environment

Smart Manufacturing Environments constitute one of the most mature and well-regulated High-Stakes domains for deploying Machine Learning. They can integrate advanced sensing technologies, robotics, and automation systems to optimize the operation of industrial plants. Enabling real-time monitoring, adaptive control, and predictive analytics, these environments aim to improve productivity, minimize unplanned downtime, and enhance quality assurance processes. [1]

[1] This chapter is based primarily on the following authors' publications:

- R. Venanzi, S. Dahdal, et al., "Enabling adaptive analytics at the edge with the bi-rex big data platform," Computers in Industry, vol. 147, p. 103 876, 2023.
- L. Colombi, M. Vespa, N. Belletti, et al., "Embedding models for multivariate time series anomaly detection in industry 5.0," Data Science and Engineering, Jun. 2025.
- S. Dahdal, L. Colombi, M. Brina, et al., "An mlops framework for gan-based fault detection in bonfiglioli's evo plant," Infocommunications journal, vol. 16, no. 2, pp. 2–10, 2024.
- L. Colombi, A. Gilli, S. Dahdal, et al., "A machine learning operations platform for streamlined model serving in industry 5.0," in NOMS 2024-2024 IEEE Network Operations and Management Symposium, 2024.
- L. Colombi, M. Vespa, N. Belletti, et al., Multivariate time series anomaly detection in industry 5.0, 2025. arXiv: 2503.15946 [cs.LG].

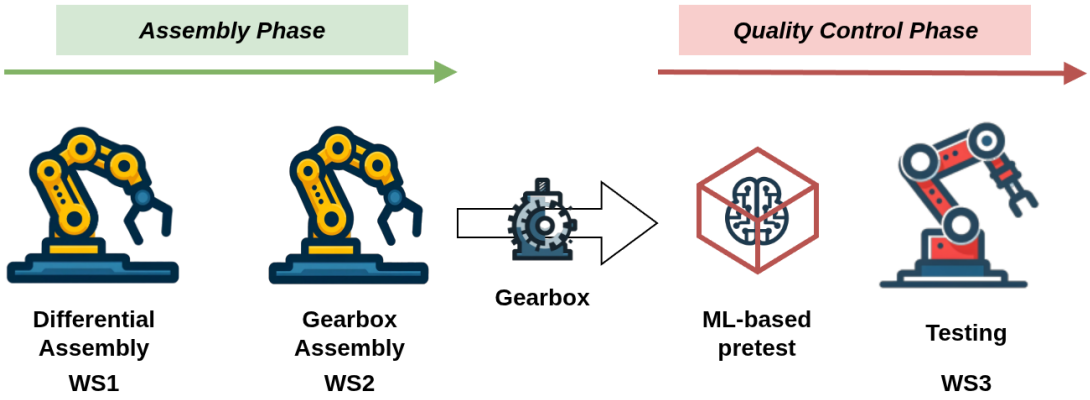


Figure 5.1: Bonfiglioli gearbox assembly line at the EVO plant.

5.1 Use Case of Bonfiglioli Riduttori

Bonfiglioli Riduttori[2] is a global leader in the design and manufacturing of gearmotors, drive systems, planetary gearboxes, reducers, and inverters. The company has been at the forefront of embracing Industry 5.0 principles, placing great emphasis on efficiency, innovation, and environmental sustainability within its production processes. Its manufacturing facilities include a diverse range of machines, such as the *Gleason P800* gear hobbing machine and large-scale industrial gearbox manufacturing plants, including the *EVO Plant*. These facilities form the basis of our case study, serving as a clear example of a Type 1 High-Stakes Environments. They provide an ideal context to examine how advanced Machine Learning solutions can be integrated into established industrial workflows to enhance productivity, ensure quality, and support sustainable manufacturing practices.

- L. Colombi, M. Brina, M. Vespa, et al., "Optimizing industry 5.0 machine learning-based applications via synthetic data generation," in 2024 IEEE 29th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD), 2024.
- L. Colombi, I. Boleac, M. Brina, et al., "Multi-cluster mlops platform for industry 5.0," in 2025 IEEE Symposium on Computers and Communications (ISCC), IEEE, 2025.

[2] https://www.bonfiglioli.com

Gear Hobbing Machine

The gear manufacturing process typically begins with fusion, followed by stamping, turning, and spline cutting. These are succeeded by external processing steps, such as hardening and carburizing, and then by internal finishing operations, including diameter grinding and tooth grinding. In the first part of this study, we will focus on the gear-cutting phase, performed by the *Gleason P800*, a precision hobbing machine specifically designed for pinion shafts with external gearing. The machine employs a disk cutter with replaceable inserts to execute rough-cut operations with high accuracy and repeatability. The pinions produced in this phase are primarily destined for gear reducers used in the wind energy sector, where durability, dimensional precision, and surface quality are critical for operational reliability.

Gearbox Assembly Plant

In parallel, the second part of this study focuses on a production gearbox assembly and testing line at the *EVO Plant*, which serves as a particularly relevant example. This line integrates advanced automation technologies to enable high-precision assembly while ensuring rigorous quality assurance.

The assembly process is organized into three primary workstations (WS), as illustrated in Figure 5.1: **WS1 - Differential Assembly**: Responsible for assembling the gearbox's differential unit. Key process data, including insertion forces and torque values during tightening operations, is recorded in real time. **WS2 - Gearbox Assembly**: Completes the full gearbox assembly while collecting similar process data, mirroring WS1 in capturing insertion forces and torque metrics for traceability and process optimization. **WS3 - End-of-Line Testing**: Performs a comprehensive evaluation of the assembled gearbox using two specialized test machines: *Performance Validation Machine*, executes stress tests comprising two static runs at 800 revolutions per minute (RPM) and two ramp-up sequences reaching 11000 RPM, assessing the component's

performance under varying operational loads. *Vibration Analysis Machine*, Monitors vibrational behavior throughout the testing process, capturing vibration data at increasing RPM thresholds to evaluate structural integrity and dynamic stability under simulated real-world conditions.

5.1.1 Goal and Requirements

For the *Gleason P800* gear hobbing machine, the objective is to design an Machine Learning system capable of detecting anomalies throughout the gear-cutting process. This system will leverage the various operational metrics collected by the machine to identify deviations from expected patterns, enabling early fault detection, reducing scrap rates, and improving process stability.

For the *EVO* gearbox assembly line, the primary goal is to enhance efficiency, product quality, and sustainability through the integration of a *pretesting phase* between WS2 (Gearbox Assembly) and WS3 (End-of-Line Testing). This additional checkpoint serves as an early-stage quality control step, aimed at identifying defective components before they reach the final testing phase, which is significantly more resource-intensive and costly. To successfully develop the ML system, several technical challenges and requirements must be addressed:

1. Reliable Classification on Imbalanced Data The system must accurately distinguish defective from non-defective units using process data collected from WS1 and WS2. The dataset exhibits a strong class imbalance, with defective cases representing only a small fraction of total production. Developing a model that maintains high sensitivity (recall) for defect detection while preserving specificity under these conditions is a critical requirement.

2. Anomaly Detection on Time Series Hobbing Data: The *Gleason P800* produces high-resolution time series data capturing metrics such as insertion forces, torque values, and other assembly parameters. An effective solution must leverage these sig-

nals to detect anomalous patterns indicative of early-stage faults or process deviations. This requires Anomaly Detection techniques that can model temporal dependencies and identify subtle deviations from normal operating behavior.

3. Robust MLOps Infrastructure: A production-grade MLOps platform is essential to manage the entire lifecycle of deployed models. This includes capabilities for model deployment, continuous performance monitoring, automated retraining, version control, and controlled rollouts of updates.

Together, these requirements underline the need for a holistic Machine Learning solution that combines high predictive accuracy with seamless integration into the operational and computational infrastructure of the manufacturing environment.

5.2 Dataset Collection and Preprocessing

This section details the procedures adopted for acquiring, preprocessing, and balancing the datasets used in this study. Two primary datasets are considered: (1) the dataset for developing the Anomaly Detection model for the Gleason P800 hobbing machine and (2) the dataset for training the WS3 pretesting classification model. For each case, the data collection methods, preprocessing steps, and class balancing strategies are outlined to ensure the resulting datasets are representative, reliable, and suitable for model training and evaluation.

5.2.1 Hobbing Machine Data

The data collected from the Galeson P800 extensive Time Series data from various sensors installed on the machine during the gear-cutting process. Each monitored Bonfiglioli manufacturing process was recorded for different cutting periods and gears, for a total of one million rows. Each row represents a specific time instance, and for each second, 104 different measurements were taken. Therefore, each multivariate time series (MTS)

was originally made of 104 dimensions.

Next, we checked the presence of NaN values and deleted duplicate rows and outliers. The forward-filling method was employed to substitute NaN values with the value from the previous row. Regarding outliers, it was important to remove them due to the unlabeled nature of the dataset, which may contain noise or anomalies. To reach this goal, we followed the advice of the domain experts and deleted rows where at least one value was below the first or above the third quintile, calculated on the single column. Some files contained only 90 rows, while others contained over 10,000 due to the varying duration of the manufacturing process. Since data are collected every second, we had to manage observation files spanning from a minimum of 90 seconds to a maximum of 160 minutes.

The dataset was processed and split into multivariate fixed-size Time Series of 100 seconds. The samples shorter than 100 seconds were extended by applying constant padding, specifically prolonging the last value. Followed by the selection of 6 key features from the original 104 measurements in the Bonfiglioli dataset, which was driven by domain experts' requirements. These specific measurements capture the load on various movable components within the system, including the piece holder table, the spindle, and the radial, tangential, and axial slides, which are responsible for the movement along different axes. Concentrating on this subset of the available data allowed us to tackle the inherent challenges of MTS analysis. After preprocessing and cleaning, the final dataset consisted of 2,950 6-features multivariate time series, with each one spanning 100 seconds.

The dataset was partitioned into training and test subsets using a 90/10 split. To thoroughly evaluate the performance of Anomaly Detection models under realistic industrial conditions, we augmented the test set. The augmented test sets were generated by (i) injecting *synthetic anomalies* and (ii) *extraneous noise* into the MTS data. Two types of synthetic anomalies were introduced, guided by domain expert input to

reflect common real-world disturbances in industrial machinery. The first type-step anomalies involved abrupt and sustained level shifts in one or more signal channels. The second type-periodic spike anomalies-was created by superimposing sequences of transient spikes, with amplitudes sampled from a three-component Gaussian Mixture Model. These anomalies simulate typical mechanical disruptions, such as sudden load changes and periodic mechanical chatter, frequently encountered in industrial environments. To assess the robustness and precision of the detection models, we also injected random *extraneous noise* into a subset representing 10% of the original data. This included single-point impulse noise and salt-and-pepper noise, simulating transient, non-informative disturbances unrelated to actual mechanical faults. This noise scenario was intentionally designed to test the models' ability to distinguish between genuine anomalies and irrelevant, sporadic perturbations.

As a result of this augmentation, the final test set was established. The labeled test sets comprise both synthetic anomalies and extraneous noise applied to the four features exhibiting the greatest variance; the remaining two features were comparatively stable, and the introduction of anomalies there would have rendered detection trivial. This systematic augmentation strategy enables a rigorous evaluation of Anomaly Detection algorithms, assessing their capacity to accurately identify true anomalies in complex and noisy industrial data settings.

5.2.2 EVO Plant Data

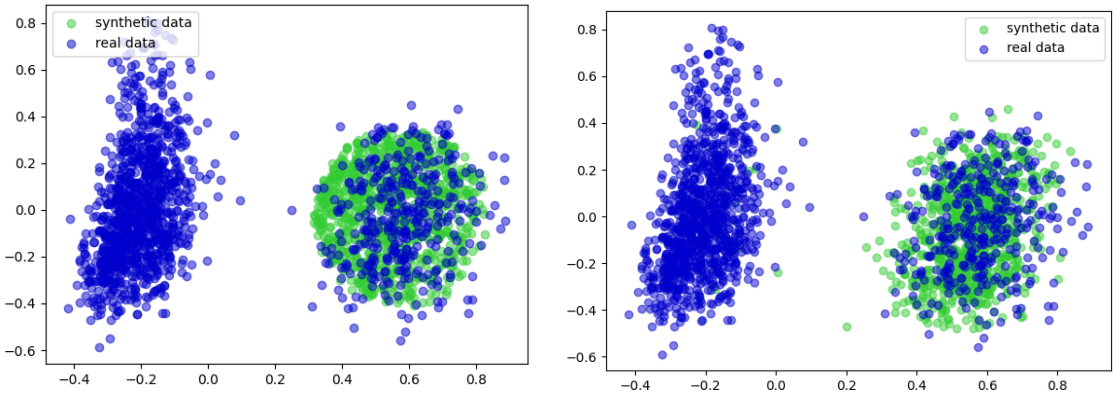
The dataset for this study was acquired from two key workstations in the EVO gearbox assembly line: WS1 (Differential Assembly) and WS2 (Gearbox Assembly). Data collection focused on two primary process variables, tightening torques and press-fit forces, both of which are directly linked to the structural integrity and operational performance of the final gearbox units.

The raw dataset initially comprised approximately 700 metrics (features) recorded

during these assembly processes. In collaboration with Bonfiglioli domain experts, a structured feature selection and preprocessing workflow was implemented to identify the most relevant variables for quality prediction. This process reduced the feature set to 66 critical variables with the strongest influence on gearbox reliability. These features were then paired with labeled outcomes from WS3 (End-of-Line Testing) to form the ground truth for the ML-based pretesting classifier. Comprehensive preprocessing steps were applied, including data cleaning and outlier removal. At this point, class distribution analysis revealed a pronounced imbalance, with approximately 85% of instances labeled as "good" and only 15% as "broken" (faulty). This imbalance reflects the high yield and precision of Bonfiglioli's manufacturing processes but introduces challenges for model training, as it can bias predictions toward the majority class and reduce defect detection sensitivity. The final dataset contained 1375 gearbox instances, partitioned into an 80/20 train-test split:

- **Train set:** 1,100 instances (935 good, 165 broken)
- **Test set:** 275 instances (234 good, 41 broken)

To address the class imbalance, synthetic data generation techniques based on Generative AI were applied to the training set to achieve a balanced representation of both classes, thereby improving the robustness and generalization capability of the resulting classifier. The first approach employed was the Conditional Tabular GAN (CTGAN) [31], a generative model specifically designed for conditional tabular data synthesis. The second approach used was the Wasserstein Generative Adversarial Network (WGAN) [33], combined with a cluster-based filtering method as described in [20], to selectively generate and retain synthetic samples of faulty gearboxes. Fig. 5.2 illustrates the relationship between the real and synthetic datasets for both methods. For visualization, the dataset was first cleaned and then reduced from 66 features to two dimensions using Principal Component Analysis (PCA). The PCA transformation revealed two distinct clus-



(a) WGAN (b) CTGAN

Figure 5.2: Visualization of the augmented datasets generated by (a) WGAN and (b) CTGAN, after dimensionality reduction to two principal components using PCA. Blue points represent real data samples, while green points correspond to synthetic samples generated for class balancing.

ters: a denser left cluster representing "good" gearboxes and a right cluster representing "broken" gearboxes. In both the WGAN- and CTGAN-generated datasets, synthetic data points are overlaid with the original data, enabling a qualitative comparison of distributional similarity. The CTGAN-generated samples most closely replicate the structure of the original data, successfully reproducing the two-cluster separation observed in the real dataset.

Secondly, to obtain a more precise and quantitative comparison of the two GAN models, we employed a set of distance and similarity metrics designed to characterize differences in data distributions. These metrics were chosen for their ability to capture both global and local distributional properties, providing a comprehensive assessment of generative performance. The selected measures were: Wasserstein Distance (WD), the complement of the Kolmogorov-Smirnov (KS) D statistic, Correlation Similarity (KS-C), and Jensen-Shannon (JS) Distance [20]. First, the *Wasserstein Distance* (WD) was employed to assess the similarity between the original and synthetic datasets, with smaller values indicating greater similarity. The second metric, the *Kolmogorov–Smirnov D*

statistic, measures the maximum difference between two empirical cumulative distribution functions. A larger D value indicates a more significant discrepancy between the distributions of the real and synthetic gearbox data. The KS D statistic is derived from the KS test, which was applied to compare the probability distribution characteristics of features from the actual and generated datasets. Additionally, the *KS-Complement* was used as a supplementary indicator, defined as $1 - D$, providing an alternative representation of distributional similarity. The *Correlation Similarity* metric was also computed to evaluate the relationships between pairs of numerical variables. It compares pairwise correlations in the real and synthetic datasets, where values closer to 1.0 indicate stronger agreement in correlation structure. Finally, the *Jensen–Shannon distance* was calculated as another measure of divergence between two probability distributions.

A summary of the metric results for both GANs is reported in Table 5.1. The findings indicate that CTGAN achieved marginally better scores across most metrics, corroborating the earlier visual analysis and suggesting that it produces synthetic data distributions more closely aligned with the real dataset.

Table 5.1: Performance comparison of WGAN and CTGAN based on distribution similarity and distance metrics. KS-C (higher is better), CS (higher is better), WD (lower is better), JS (lower is better). Higher similarity scores and lower distance scores indicate a closer match between the synthetic and real data distributions.

	KS-C	CS	WD	JS
WGAN	0.759	0.922	0.024	0.309
CTGAN	0.799	0.931	0.022	0.261

5.3 Machine Learning Model Training

After preparing and balancing the datasets, the next stage involves training the Machine Learning models for both the WS3 pretesting classifier and the hobbing Anomaly De-

tection system.

5.3.1 Anomaly Detection

Anomaly Detection plays a critical role in maintaining operational efficiency, enabling predictive maintenance, and preventing costly equipment failures. In the context of the hobbing process, it serves as an early warning mechanism by identifying deviations from expected operational behavior before they escalate into major issues. A central challenge in industrial Anomaly Detection is the occurrence of previously unseen faults. Such cases limit the applicability of supervised learning approaches, which require labeled examples of all possible failure modes. Furthermore, the analysis of multivariate time series (MTS) data introduces additional complexity, as it requires simultaneous modeling of multiple sensor signals and their interdependencies over time.

To address these challenges, we focus on unsupervised and self-supervised learning approaches that can detect anomalies without prior knowledge of every possible fault type. Among these, autoencoder (AE)s have emerged as a powerful technique for multivariate time series analysis [68]. By learning compact embeddings that capture the underlying structure of the normal operational state, autoencoders can reconstruct expected behavior and flag significant reconstruction errors as potential anomalies.

A novel framework was developed that combines a Time2Vec-inspired [69] autoencoder architecture with various Machine Learning algorithms for Anomaly Detection. The original Time2Vec method was introduced to produce a richer representation of time by mapping a scalar time input into a higher-dimensional vector, and was primarily designed for univariate time series data. However, the temporal dimension is inherently embedded within the MTS values themselves, enabling us to replace the explicit scalar time input with the observed time series features. Building on this idea, we extend the Time2Vec approach beyond its original univariate formulation, developing a method to generate embeddings directly from MTS data. This extension enables the representa-

tion of both temporal dynamics and inter-feature relationships in a unified embedding space. Let X denote an MTS of size $N \times F$, where N is the number of time steps and F is the number of features. Given an embedding dimension K (a tunable hyperparameter), we define the function $T2V_{MTS}(X)$ to produce a matrix representation, which is subsequently flattened into a vector. Each column i of this embedding is computed as:

$$T2V_{MTS}(X)[i] = \begin{cases} X\,\omega_0 + b_0 & \text{if } i = 0 \\ \sin\left(X\,\omega_i + b_i\right) & \text{if } 1 \leq i < K \end{cases} \tag{5.1}$$

In this formulation, the $\sin(\cdot)$ function operates element-wise. The parameter w_0 is a weight vector of dimension $F \times 1$, and b_0 is a bias vector of size $N \times 1$. For $i \geq 1$, w_i denotes weight matrices of dimension $F \times (K-1)$, while b_i represents bias matrices of size $N \times (K-1)$. This design enables the embedding to simultaneously capture both non-periodic patterns $(Xw_0 + b_0)$ and periodic patterns $(\sin(Xw + b))$. By employing the sine transformation, the embedding is also made invariant to time rescaling effects [70]. The output of the $T2V_{MTS}$ layer is then flattened into a one-dimensional vector, which serves as a compact representation of the original multivariate time series. In the autoencoder architecture, the decoder reconstructs the original time series from this embedding using a reshaping layer followed by a sequence of 1-D convolutional layers. During the training of the autoencoder, the reconstruction error was quantified using the L2 loss. To detect anomalies it was employed the learned embeddings were employed in combination with Machine Learning algorithms to perform binary anomaly classification. Specifically, we evaluated three state-of-the-art one-class classification algorithms [71], [72]: Local Outlier Factor (LOF), Deep Isolation Forest (D-IF), and One-Class Support Vector Machine (OC-SVM).

To provide a meaningful performance baseline for our custom-designed Time2Vec-inspired autoencoder models, we applied the same optimization, training, and evaluation procedures to a standard convolutional autoencoder architecture. This baseline AE

adopts a mirrored encoder-decoder structure, where both components comprise a sequence of Conv1D layers followed by a flattening layer and a dense bottleneck layer. Our evaluation addressed two main objectives.

First, we aimed to directly compare the performance of the reconstruction-error-based Anomaly Detection approach against the embedding-based classification approach. A reconstruction error threshold is determined as the maximum reconstruction error observed in the training set. This error is computed using three complementary loss functions: Dynamic Time Warping (DTW), Mean Squared Error (MSE), and L2 norm. A time series instance is classified as anomalous if any of its three reconstruction error values exceed the corresponding threshold.

Second, we sought to assess the quality of the embeddings generated by our convolutional Time2Vec-inspired AE relative to those produced by other AE architectures. The specific choice of a convolutional AE baseline was selected due to its prevalence and demonstrated effectiveness in the literature, ensuring a fair and representative benchmark [73]. This choice aligns with standard practice in the Anomaly Detection literature, where convolutional autoencoders are frequently employed as reference models for evaluating new architectures.

Table 5.2: Optimized Autoencoders hyperparameters for the Bonfiglioli dataset

Model	**Embedding Dimension**	**Depth**	**Kernels**	**Channels**
Baseline AE	18	1	[4]	[6]
Ad-hoc Time2Vec AE	13	2	[7, 3]	[6, 35]

The results in Table 5.3 demonstrate that the Time2Vec-based (T2V) embeddings consistently outperformed the reconstruction-based convolutional autoencoder across all evaluated Machine Learning classifiers. For all three algorithms, Deep IF, One-Class SVM, and LoF, T2V achieved higher F_1 and $F_{0.5}$ scores, as well as improved precision, while maintaining perfect recall in most cases. These findings indicate that the T2V

Table 5.3: Comparison of ML models using different metrics on T2V and AE

ML	**T2V**				**AE**			
	F1	$F_{0.5}$	P	R	F1	$F_{0.5}$	P	R
Recon. Error	0.78	0.69	0.64	1.00	0.12	0.10	0.09	0.16
Deep IF	0.77	0.69	0.64	1.00	0.62	0.50	0.45	1.00
One Class SVM	**0.84**	**0.77**	**0.72**	**1.00**	0.41	0.30	0.26	1.00
LoF	0.77	0.69	0.64	1.00	0.63	0.66	0.69	0.58

embedding approach yields richer and more discriminative feature representations for Anomaly Detection compared to reconstruction-error-based methods. In particular, the One-Class SVM paired with T2V embeddings achieved the highest overall performance ($F_1 = 0.84$, $P = 0.72$, $R = 1.00$), highlighting the potential of the proposed embedding framework to enhance detection accuracy in high-stakes industrial monitoring scenarios.

5.3.2 WS3 Pretest Classifier

For the WS3 pretesting task, a Logistic Regression (LogReg) model was selected and trained on the enriched and balanced dataset. LogReg was chosen for its inherent interpretability, a critical factor in manufacturing applications where decision-making processes must be transparent and interpretable. Three versions of the classifier were developed:

- Baseline: trained on the original training dataset without augmentation.
- WGAN augmented: trained on a dataset augmented with synthetic samples generated by WGAN and refined through unsupervised cluster-based filtering.
- CTGAN augmented: trained on a dataset augmented with synthetic samples generated by a conditioned CTGAN model.

While CTGAN achieved superior results in similarity-based metrics in 5.1, the WGAN with filtering produced more effective training outcomes. This advantage is attributed to

Table 5.4: LogReg Model performance metrics.

Metric	Baseline	WGAN augmented	CTGAN augmented
Accuracy	0.90	0.91	**0.92**
Precision	0.65	0.64	**0.66**
Recall	0.81	**1.00**	0.95
F1 Score	0.72	**0.78**	**0.78**
F2 Score	0.77	**0.90**	0.88
G-Mean	0.86	**0.95**	0.93

WGAN's ability to better manage mislabeled data points present in the original dataset. In contrast, CTGAN augmentation sometimes amplified the problem by generating additional mislabeled samples, thereby introducing bias and noise into the training process. The WGAN augmented model notably reduced the number of false negatives, cases where broken gearboxes were incorrectly classified as functional, and improved key performance indicators such as recall, F_2 score, and geometric mean (G-mean). These improvements enhance the model's ability to reliably identify defective units, a critical requirement under a zero-waste manufacturing paradigm. In conclusion, despite CTGAN's comparable performance in distribution similarity metrics, WGAN proved to be the more suitable augmentation strategy for this specific application, delivering tangible gains in fault detection reliability.

5.4 MLOps Platform Deployment

Deploying Machine Learning models and applications in industrial environments requires managing the entire machine learning life cycle, from data acquisition to operational integration. In this work, this process begins with the adoption of the Big Data-oriented *Bi-Rex Platform* for large-scale data ingestion, storage, and processing. This is followed by the use of the *NGA4M Platform*, which provides the necessary tools and infrastructure to manage the machine learning life cycle, including model training, eval-

uation, deployment, and monitoring. Together, these platforms enable end-to-end life-cycle management, ensuring seamless integration from initial data collection through to the deployment of production-ready ML models in high-stakes manufacturing contexts.

5.4.1 Bi-Rex Platform

The first deployment phase focused on integrating data collection, preprocessing, and basic foundational MLOps components, with particular emphasis on IT/OT convergence. This integration was achieved through the *Bi-Rex Platform*, developed within two PR-FESER regional projects, *BD4M* and *DEEPMON*, in collaboration with multiple industrial partners [63]. As shown in Figure 5.3, the DEEPMON platform manages the Edge/OT layer, while BD4M provides Cloud/IT layer services, together enabling an end-to-end, cloud-to-edge DevOps and MLOps workflow.

DEEPMON

At the OT level, DEEPMON provides secure connectivity and communication between industrial devices, along with capabilities for data standardization, enrichment, and temporary caching to ensure data integrity before transmission. Connectivity is achieved through an OPC-UA connector that standardizes machine-specific protocols, with MQTT delivering standardized data to edge microservices. A Kafka-based messaging service enables reliable OT-IT communication, while an ETL module integrates and enriches incoming data before persisting it in a MongoDB-based storage service. This design supports real-time visualization, local ML model execution, and the deployment of new services or configurations directly from the IT layer, reducing latency, preventing error propagation, and enabling agile runtime updates.

At the IT level, BD4M offers functionalities for large-scale data ingestion, long-term storage, and distributed processing, facilitating advanced analytics and Machine Learning workflows. Additionally, it incorporates visualization tools that allow stakeholders

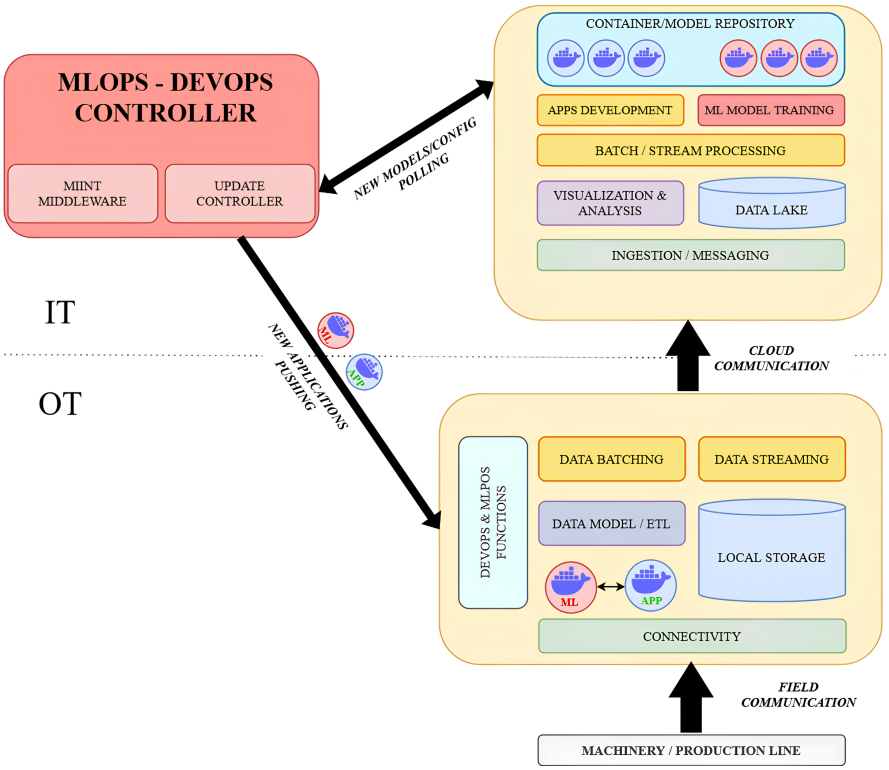


Figure 5.3: Logical architecture of Bi-Rex platform for cloud-to-edge DevOps and MLOps functions.

to monitor processes, track system performance, and interact with analytical results in real time. It ingests data from heterogeneous sources, including sensors, remote storage, cloud services, and other edge nodes, using Kafka and MQTT brokers for scalable, publish-subscribe communication. An internal data lake provides long-term, centralized storage and a unified access point for downstream applications. BD4M also supports distributed data processing and validation across multiple abstraction levels (sensor, single machine, production line, or multi-plant) via querying. Integrated visualization and decision-support tools allow end users to interact with both raw and processed data through customizable web dashboards, enabling the creation of comprehensive analytics and ML applications directly within the platform.

Together, DEEPMON and BD4M form a tightly coupled edge-to-cloud ecosystem that ensures high-quality, standardized data collection, robust storage, and scalable processing for industrial settings.

MLOps-DevOps Controller Automatic Deployment

The MLOps-DevOps Controller (MDC), positioned at the IT layer, enables automated monitoring, configuration, and deployment of services and ML models to DEEPMON edge nodes. Integrating DevOps and MLOps practices, MDC manages the full deployment lifecycle in a platform-agnostic manner. It comprises two main modules: the Update Controller (UC), which monitors the BD4M container repository for new service or model versions, often triggered by performance drift alerts, and automatically deploys them to the edge; and the MIINT Middleware, a microservice-based SOA layer that abstracts platform-specific details through standardized REST APIs, enabling dynamic updates across heterogeneous OT environments, including Siemens Industrial Edge. A built-in container and model registry ensures reliable storage of Docker images and ML models. By automating orchestration, updates, and configuration changes, MDC delivers full DevOps capability to the OT layer, making edge operations agile, scalable, and

suitable for SME manufacturing contexts.

5.4.2 NGA4M Platform

The NGA4M platform automates and manages the complete machine learning life cycle, from data preprocessing to model production deployment, while ensuring reproducibility, scalability, and resilience in dynamic industrial environments. It integrates MLOps activities such as data processing, model training, evaluation, deployment, and continuous monitoring with DevOps practices including CI/CD, version control, and automated orchestration. As shown in Figure 5.4, the platform supports seamless integration between IT systems and edge devices, enabling unified data flow, infrastructure orchestration, and dynamic resource provisioning across edge, private, and public clouds. Built-in service management functions, such as auto-scaling, load balancing, and self-healing, maintain operational efficiency in real time.

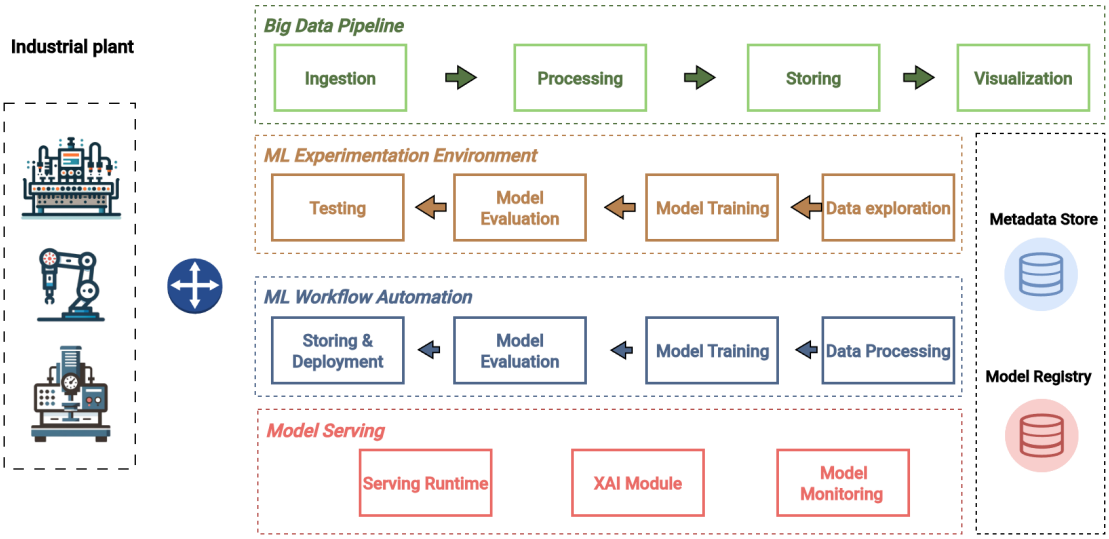


Figure 5.4: MLOps NGA4M Platform

ML Experimentation Environment:

Provides data scientists with tools for data exploration, model training, evaluation, and testing. It connects to distributed storage layers (data lakes, warehouses, feature stores) and leverages a Model Registry and Metadata Store to track experiments, versions, and configurations, ensuring repeatability and traceability. Integration with CI/CD pipelines allows automated triggering of workflows upon code or data changes.

ML Workflow Automation:

Bridges experimentation and production by orchestrating data processing, training, evaluation, and deployment. Workflows can be triggered by code changes, model drift detection, or manual actions, enabling scalable ML operations in real-time industrial settings.

Model Registry & Metadata Store:

Centralizes storage and governance of all models, maintaining version history, performance metrics, and descriptive metadata. The Metadata Store logs experiment details, facilitating debugging, collaboration, and compliance.

Model Serving & Monitoring:

Deploys production-ready models through APIs supporting low-latency, high-throughput, and fault-tolerant inference across edge and cloud. Integrates XAI [74] (Explainable AI) to provide interpretable predictions, alongside monitoring tools for both service-level metrics (e.g., latency, resource usage) and model-level performance (e.g., drift, accuracy). Supports Continuous Training (CT) to adapt models to evolving production conditions without disrupting operations. The NGA4M platform's design presents itself as a combination of automation, integration, and explainability, making it particularly suited for any smart manufacturing as a representative Type 1 High-Stakes Environments.

5.5 Chapter Summary

This chapter examined Type 1 Smart Manufacturing Environments as a mature, well-regulated High-Stakes domain for deploying Machine Learning. We explored the Bonfiglioli Riduttori case study illustrated the practical integration of ML into industrial workflows, with two focal points:

Gleason P800 gear hobbing machine for multivariate time series Anomaly Detection and the EVO gearbox assembly line for early defect classification via an ML-driven pretest phase. The dataset collection and preprocessing section detailed feature selection, cleaning, balancing via GAN-based data augmentation, and domain-expert validation. In model training, we introduced a novel Time2Vec-based autoencoder for multivariate time series Anomaly Detection, outperforming baseline reconstruction-based and baseline convolutional autoencoder methods. Additionally, showcasing the process of developing a Logistic Regression classifier for pretesting, where WGAN augmentation yielded the best recall and F2-score for defect detection. Finally, the MLOps platforms deployment describes the integration of Bi-Rex and NGA4M platforms, enabling end-to-end ML life cycle automation across IT/OT layers.

This concludes the exploration of the Type 1 High-Stakes Environments, demonstrating how smart manufacturing environments provide robust conditions for advanced ML integration. The insights and results obtained here set the stage for addressing progressively more challenging High-Stakes Environments in the following chapters.

Chapter 6

Type 2: In-Field Industrial Environments

In-field industrial environments are relatively common yet underexplored domains for deploying Machine Learning solutions. These settings typically involve machinery equipped with advanced sensors, robotics, and limited automation, enabling small- to mid-scale industrial operations. By integrating predictive analytics and intelligent control, such systems can improve quality assurance, enhance operational efficiency, and support proactive maintenance strategies [1].

[1] This chapter is based primarily on the following authors' publications:
- F. TASSI, R. Lazzarini, E. BELLODI, S. DAHDAL, C. STEFANELLI, and M. TORTONESI, Machine for making liquid, semiliquid or semisolid food products and related control method, US Patent App. 18/768,782, Jan. 2025.
- Filippo Tabanelli, Simon Dahdal, et al. «Smart and Sustainable Ice Cream Making Through Edge Machine Learning». In: IEEE Transactions on Industrial Informatics (2026), pp. 1–10.

6.1 Use-case of Carpigiani

Carpigiani, a global leader in ice cream and gelato equipment and machines, holds approximately 35% of the world market with more than 140,000 machines installed worldwide. With its reputation for reliability and performance, the company introduced a remote monitoring and diagnostics system, the Teorema e-maintenance platform [75]. First deployed in 2008, Teorema now operates across all Carpigiani machines. Teorema collects, processes, and stores diverse machine-generated data in a centralized, cloud-based infrastructure. At its core is a big data analytics pipeline designed for scalable and efficient data storage. This centralized approach ensures continuous global access to machine performance data, enabling real-time monitoring and diagnostics. By detecting anomalies and predicting failures, Teorema supports proactive maintenance: it minimizes unplanned downtime, extends machine lifespan, and ensures service interventions are carried out exactly when required.

Carpigiani's machines generate a large amount of data during operations. This data can be categorized into three primary types. *Technical data*: This includes metrics such as temperatures, motor status, mixture levels, etc. *Operational data*: This refers to information about the completion of pasteurization cycles, alarm statuses, and other machine alerts. *Production data*: This encompasses information such as the number of portions served, cycles performed, and hours of motor operation. Together, these data provide comprehensive and real-time insights into the status of a machine and its production processes.

6.1.1 Goal and Requirements

The primary goal is to develop an automated human error correction system capable of detecting whether an operator has loaded an unbalanced ice cream mixture and taking corrective action before the preparation process is compromised. Achieving this requires

fast, reliable, and local decision-making, which poses unique technical and architectural challenges. While Carpigiani's Teorema platform has proven effective for centralized monitoring, this architecture is unsuited for low-latency real-time error detection and correction. Mixture imbalances demand high responsiveness, which is incompatible with cloud communication latency. Beyond timing constraints, communication overhead must also be considered. Carpigiani machines are deployed globally, operating in diverse environments, from regions with robust 5G to areas reliant on GSM/GPRS networks, where high latency and limited bandwidth make frequent data transfer impractical. Transmitting process-specific data would further raise privacy and security concerns, as it involves sensitive production details not currently handled by Teorema. Consequently, any effective solution must rely on edge AI, where Machine Learning (ML) inference is performed directly on the machine. This ensures low-latency responsiveness and independence from unstable connectivity. However, edge devices face constraints in computational power, memory, and energy resources [76]–[78]. To be viable, deployed models must be lightweight yet capable of real-time inference, requiring optimization techniques such as quantization, pruning, and efficient architecture design. Managing the balance of accuracy and resource efficiency while avoiding costly hardware upgrades.

A final challenge is to handle dynamic classification. The system must continuously assess mixture states throughout the production cycle and detect imbalances in near real-time to allow effective correction. These rules out many standard neural network classifiers and instead demand innovative, task-specific model designs tailored for resource-constrained, real-time industrial environments, which is a perfect representation of a Type 2 High-Stakes Environments.

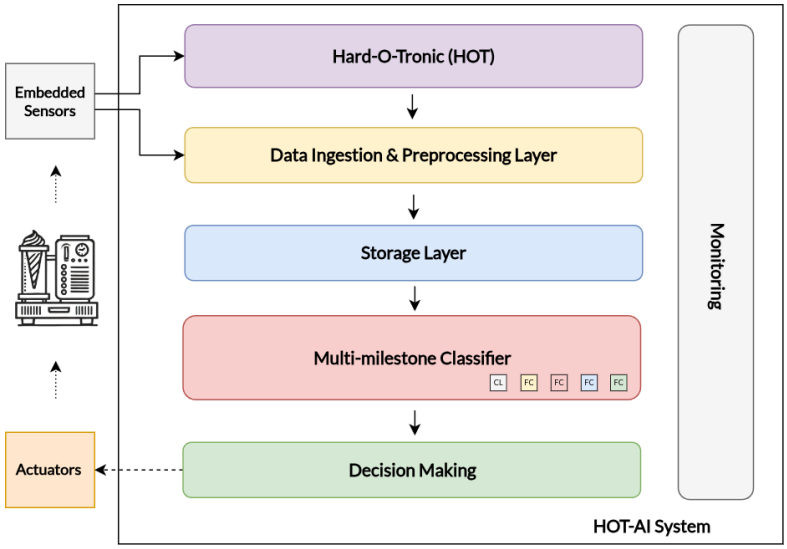


Figure 6.1: Architecture of the HoT-AI System.

6.2 The HoT-AI System

The Hard-O-Tronic AI-driven (HOT-AI) system, presented in [79], is an intelligent, fully automated error correction framework designed to operate without operator supervision. Running in near real-time, it continuously monitors the ice cream production process to detect unbalanced ingredient mixtures and dynamically adjusts preparation parameters to minimize waste and ensure consistent product quality.

System Design

A high-level overview of the HoT-AI architecture is shown in Fig. 6.1. At the core is the Monitoring component, which is triggered by the machine's HMI subsystem when an operator initiates a production cycle. This component supervises the entire HoT-AI workflow, orchestrating communication among system modules.

The workflow begins with a continuous stream of sensor data (e.g., temperature, pressure, viscosity) captured by the machine. This raw data is passed to the Hard-O-Tronic (HOT) system, a proprietary Carpigiani technology that applies advanced heuristic algo-

rithms to evaluate the consistency and quality of the ice cream mixture. By enriching raw sensor data with HoT-derived insights, the system produces an information-rich dataset that can reveal anomalies such as ingredient imbalances, excess sugar or fat, and deviations in water content. The enriched data is then processed through the Data Ingestion and Pre-processing Layer, where it is cleaned, normalized, and structured for analysis. Finally, the processed data is archived in the Storage Layer, enabling both real-time decision-making and long-term historical analysis.

Multi-Milestone Classifier

The Multi-Milestone Classifier performs inference on the processed data stored in the Storage Layer, identifying anomalies linked to unbalanced ingredient mixtures. Specifically, it evaluates the mixture at four preparation milestones, 150, 200, 250, and 300 seconds, as decided by the domain experts, and classifies it into one of five categories: balanced, excess cream, excess sugar, insufficient sugar, or excess water. This classifier is a pre-trained ML model; importantly, HoT-AI does not support on-device learning or fine-tuning during operation. The Multi-Milestone Classifier represents the system's most significant innovation and is analyzed in detail in the next section. When an imbalance is detected, the classifier forwards a notification to the Decision Making component. This module determines whether corrective actions are required by considering the broader production context, such as the selected recipe and the current stage of the process. For instance, imbalance alerts raised early in preparation may be less accurate and do not always justify immediate intervention. If corrective action is warranted, the Decision Making component issues actuator commands to the machine, enabling fine-grained adjustments to the preparation parameters. This feedback loop ensures that the mixture is recalibrated in time, keeping the process optimized and guaranteeing that the final gelato consistently meets quality standards.

Table 6.1: Preliminary dataset versions tested for training the Multi-Milestone Classifier. Each dataset contains 103 time series with fixed lengths ranging from 150 to 450 seconds. The percentage of generated and truncated data indicates the proportion of padded or discarded elements required to standardize sequence length. Dataset IDs follow the convention "D*time_series_length*". Where TS denotes time series.

Dataset ID	TS length (s)	Data generated	Data truncated
D150	150	0.00%	49.27%
D200	200	0.41%	32.65%
D250	250	1.81%	17.00%
D300	300	6.81%	5.41%
D350	350	16.74%	1.45%
D400	400	26.39%	0.44%
D450	450	34.29%	0.00%

6.3 Dataset Collection and Preprocessing

Training the Multi-Milestone Classifier required collecting test runs from Carpigiani machines in operation and generating training and test sets from the recorded data. Each machine provides sensor readings at one-second intervals, producing multivariate time series of variable lengths ranging from 250s to 450s, depending on the production cycle duration.

To construct a consistent dataset, we first evaluated the most suitable time series length. Seven dataset versions were created (Table 6.1), each containing 103 time series standardized to a fixed duration: 150 s, 200 s, 250 s, 300 s, 350 s, 400 s, or 450 s.

Standardization was performed by either padding or truncation. For shorter series, the last recorded element was replicated until the target length was reached; for longer series, excess elements were removed. This procedure introduced varying degrees of artificial noise (from padding) and information loss (from truncation). From this analysis, the dataset with 300 s time series, D300, was selected as the most balanced option, as it

Table 6.2: Classes of imbalance: Balanced recipe, excess of cream (Cream+), excess of sugar (Sugar+), reduction in sugar (Sugar-) and excess of water (Water+). The percentages indicate the proportion of each class in the D300 dataset.

Class of imbalance	Balanced	Cream+	Sugar+	Sugar-	Water+
Percentage	28%	24%	14%	10%	24%

minimized both generated (6.81%) and truncated (5.41%) data. This version was used for all subsequent experiments. Data was also cleaned by removing outliers and applying normalization to standardize input features. Every time series in the dataset corresponds to a production cycle of a single recipe. To label the time series (recipes) as balanced or unbalanced (compared to the balanced recipe), we referred to the five classes shown in Table 6.2:

- "*Balanced*" recipe represents the standard, correct recipe;
- "*Cream+*" indicates an increased cream content of $+$ [300-400%];
- "*Sugar+*" an increased sugar level of $+$ [33-100%];
- "*Sugar-*" a reduction of $-$ [33-67%];
- "*Water+*" an increased water content of $+$ [10-20%].

These imbalances were intentionally introduced in the test runs to mimic potential operator errors or variances in production conditions. The final D300 dataset consisted of 103 labeled time series, split into a training set of 71 series (70%) and a test set of 32 series (30%).

6.4 Model Design and Training

Within HOT-AI, mixture classification must be performed at multiple stages of the ice cream production process to provide continuous quality feedback with increasing levels

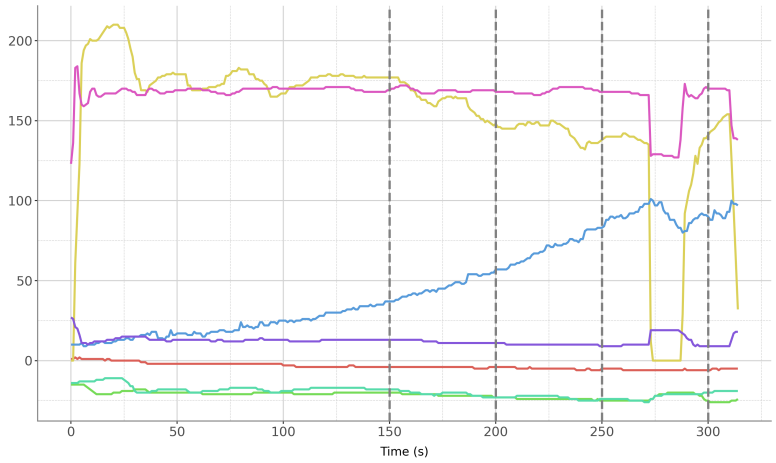


Figure 6.2: Time series collected from the machine with 4 milestones at +150s, +200s, +250s, and +300s from the start of production.

of accuracy. Consequently, the ML models must be capable of handling multivariate time series of variable lengths, depending on the elapsed time from the start of production to the classification milestone (Fig. 6.2).

Several neural architectures commonly applied to time series classification were evaluated, including CNNs, RNN, GRUs, LSTMs, and hybrid combinations. After extensive experimentation, CNN were selected due to their superior performance and computational efficiency, both essential given the resource constraints of edge devices. CNNs demonstrated strong capability for automatic feature extraction, enabling effective representation of the time series dynamics relevant to mixture quality classification.

Multi-Milestone Classifier

The CNN trained on the *D300* dataset is composed of a convolutional layer and a fully-connected layer, indicated respectively by "CL" and "FC300", as shown in Fig. 6.3. This network is capable of performing classification of new recipes at a time of +300s. As the objective was to design a multi-milestone classifier, inference needed to be performed not only at +300s but also at earlier stages (+150s, +200s, +250s). To achieve this,

three additional fully connected layers were trained, each corresponding to a shorter dataset: *D250*, *D200*, and *D150*. Each of these datasets was split into training (70%) and test (30%) sets, maintaining recipe-level consistency across splits with the same methodology applied to *D300*.

Rather than training a separate CNN from scratch for each dataset, we reused the pre-trained convolutional module obtained from training on the *D300* dataset. This convolutional layer was frozen, meaning its weights were not updated during subsequent training, as it already encapsulated the feature extraction capabilities learned from the longer time series. Only the Fully Connected (FC) layers, more sensitive to variations in input length, were retrained. This strategy ensured computational efficiency while maintaining robust feature representations. The overall approach is highlighted in Fig. 6.3, where the "CL" module is depicted with dashed lines to indicate that it is not modified. The overall approach is illustrated in Fig. 6.3, where the "CL" module is shown with dashed lines to indicate that its parameters remain unchanged. Conceptually, this method draws inspiration from Transfer Learning [80], where a model is first trained on a source domain D_s and later adapted to a related target domain D_t through selective fine-tuning. In our case, knowledge acquired from the *D300* dataset was transferred to shorter time series datasets, preserving feature consistency while tailoring the classification stage to different input lengths, where $F_{ts}(D)$ represents the set of features derived from the time series (*ts*) of dataset D:

$$F_{ts}(D150) \subset F_{ts}(D200) \subset F_{ts}(D250) \subset F_{ts}(D300)$$

where $F_{ts}(D)$ represents the feature set derived from time series of dataset D. In other words, *D150* captures only a subset of the features contained in *D200*, which in turn is nested within *D250*, and so forth. The *D300* dataset, being the most complete, provides the richest feature set, enabling efficient downward transfer of learned representations to

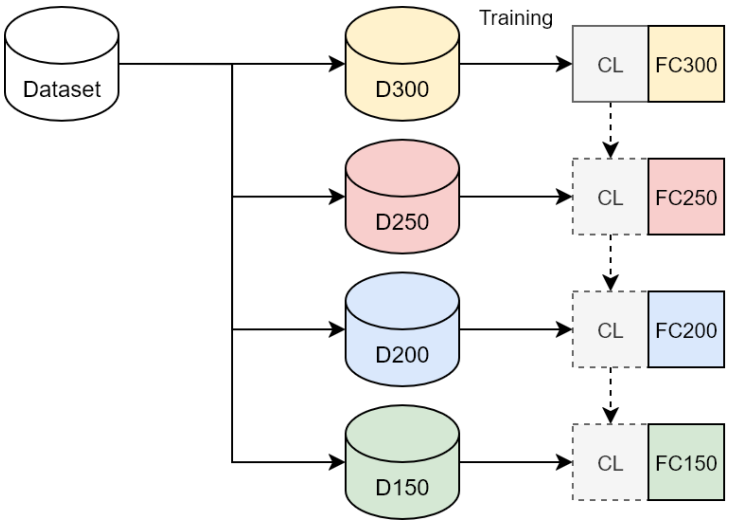


Figure 6.3: Overview of the training phase of the multi-milestone classifier, highlighting the transfer learning approach. The *D300* dataset is used for learning a CNN, composed of a convolutional layer (CL) and a Fully-Connected (FC) layer, for performing classification at milestone = 300s. The other three datasets are used for learning a specialized FC layer for performing classification at milestones M = 150s, 200s, 250s, while the CL is kept unmodified.

shorter datasets.

To validate this theoretical assumption, we empirically tested two different training strategies: a "progressive strategy" (from shorter to longer time series: 150 to 300s) versus a "reverse strategy" (from longer to shorter time series: 300 to 150). The results, reported in Tables 6.3a and 6.3b, show that the reverse strategy consistently outperformed the progressive one in terms of test accuracy across all datasets. The progressive approach, beginning with shorter sequences and gradually extending to longer ones, led to weaker feature extraction and lower accuracy when applied to extended time series. This outcome suggests that models trained on limited, short-duration inputs struggle to generalize when confronted with richer, more complex sequences. Conversely, starting from the longest dataset allowed the convolutional module to capture the full hierarchy of features, which could then be effectively transferred to shorter time series, resulting in superior generalization and performance.

Table 6.3: Accuracy (Acc.) and F1-score on training and test sets for the two training experiments: D_{300} to D_{150} and D_{150} to D_{300}. Bold values highlight the best performance.

(a) Performance Metrics for the transition from D_{300} to D_{150}.

Layer	Acc. Training Set	Acc. Test Set	F1 Score
FC300	98.59%	**96.88%**	**0.9775**
FC250	97.18%	**96.88%**	**0.9775**
FC200	94.37%	**93.75%**	**0.9525**
FC150	92.95%	**90.63%**	**0.9214**

(b) Performance Metrics for the transition from D_{150} to D_{300}.

Layer	Acc. Training Set	Acc. Test Set	F1 Score
FC150	94.37%	81.25%	0.8341
FC200	87.32%	84.38%	0.8584
FC250	98.59%	84.38%	0.8571
FC300	98.59%	87.50%	0.8795

During the training of both the convolutional and fully connected layers, hyperparameter optimization was performed using Optuna[2], a flexible and framework-agnostic optimization tool that integrates seamlessly with a wide range of machine and deep learning libraries. The search space included key hyperparameters such as the number of convolutional filters, kernel sizes, neurons per fully connected layer, learning rate, and dropout rates. This systematic optimization process enabled fine-tuning of the network architecture to achieve a balance between accuracy, computational efficiency, and robustness. As a result, the model met the real-time inference requirements of the gelato production process while remaining lightweight enough for reliable deployment on edge devices. The best-performing configuration identified for each classifier layer through this process is summarized in Table 6.4.

We compared our resulting multi-milestone classifier with a more generic, input-

[2]https://optuna.org/

Table 6.4: Architecture details for the optimized convolutional and fully connected layers in the multi-milestone classifier.

Layer	Configuration
CL	Conv1d (In: 7, Out: 128, Kernel: 6, Stride: 1) MaxPool1d (Kernel: 4, Stride: 4) Conv1d (In: 128, Out: 64, Kernel: 4, Stride: 1), MaxPool1d (4, 4)
FC300	Flatten $\rightarrow$ Linear (1088 $\rightarrow$ 74) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (74 $\rightarrow$ 86) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (86 $\rightarrow$ 5)
FC250	Flatten $\rightarrow$ Linear (896 $\rightarrow$ 31) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (31 $\rightarrow$ 62) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (62 $\rightarrow$ 140) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (140 $\rightarrow$ 5)
FC200	Flatten $\rightarrow$ Linear (704 $\rightarrow$ 128) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (128 $\rightarrow$ 42) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (42 $\rightarrow$ 26) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (26 $\rightarrow$ 5)
FC150	Flatten $\rightarrow$ Linear (512 $\rightarrow$ 96) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (96 $\rightarrow$ 35) $\rightarrow$ ReLU $\rightarrow$ Dropout (p=0.2) Linear (35 $\rightarrow$ 5)

length-agnostic alternative that integrates an Spatial Pyramid Pooling (SPP) layer [81]. This architecture consists of a Convolutional layer followed by an SPP layer and a FC classifier, i.e., $CL \rightarrow SPP \rightarrow FC$. The SPP layer addresses the challenge of variable-length time series by converting inputs of arbitrary duration into a fixed-length feature vector, thereby eliminating the need for padding or truncation. In this design, the milestone-specific FC layers of our proposed architecture are replaced with a single shared FC classifier operating on SPP-transformed features. This network was trained on combined datasets of time series of different lengths, ranging from 150 seconds to 300 seconds. To ensure a fair comparison, we used the same time series as in the previous experiments. The training and optimization procedures were kept consistent; however,

Table 6.5: Comparison in terms of accuracy (acc.) and average inference time between the multi-milestone classifier and the SPP-based architecture. In bold the highest mean values.

Milestone	Train acc. [%]		Test acc. [%]		Avg infer. time [s]	
	Ours	**SPP**	**Ours**	**SPP**	**Ours**	**SPP**
300s	98.59	89.00	96.88	85.00	0.0714	0.159
250s	97.18	87.00	96.88	85.00	0.0713	0.162
200s	94.37	85.00	93.75	85.00	0.0705	0.167
150s	92.95	86.00	90.63	82.00	0.0594	0.174
Mean	**95.77**	86.75	**94.54**	84.25	**0.068**	0.166

we introduced the *Output Size* of the SPP layer as a tunable hyperparameter, providing the layer with a flexible search space to achieve optimal performance. The optimization, performed with Optuna, identified an output size of 91×91 as the best-performing configuration. This setting provided a sufficiently rich representation to retain temporal dynamics, enabling accurate classification across variable-length inputs.

Table 6.5 reports the comparison between our multi-milestone classifier and the SPP-based alternative. Results show that the SPP model achieves lower accuracy and incurs a higher inference time across all datasets. A further key difference lies in the model size: while the SPP architecture requires 2.7 MB of memory, our solution occupies only 1.24 MB. This nearly 50% reduction in memory footprint is particularly significant for deployment on resource-constrained edge devices, such as those embedded in Carpigiani machines, where storage and processing capacity are inherently limited. Overall, the experiments demonstrate that the proposed multi-milestone classifier is not only more accurate and faster than the SPP-based model, but also more lightweight and efficient. These advantages make it a more practical and scalable solution for real-time intelligent applications in industrial machine environments.

Table 6.6: Average file size for every module of the multi-milestone classifier.

NN block	CL	FC300	FC250	FC200	FC150	Total
Size (KB)	152	344	156	382	210	1240

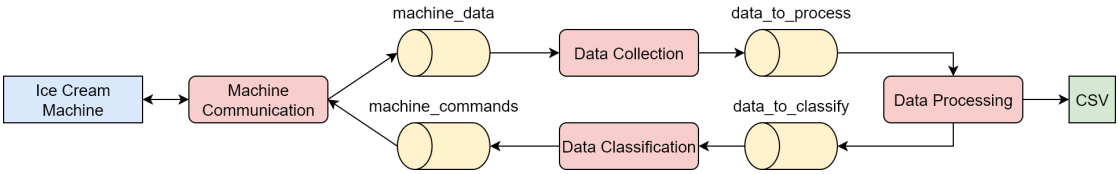


Figure 6.4: Communication Pipeline from the data source across the various modules

6.5 Model & Platform Deployment

The final platform was deployed on a Raspberry Pi 4 (2GB RAM), chosen for its cost-effectiveness, low power consumption, and sufficient processing capability for real-time classification. The total model size is only 1.24MB (Table 6.6), making it lightweight enough for constrained edge environments. Average inference times are below 0.1s, ensuring responsiveness for real-time applications.

To ensure reliable operation on resource-limited hardware, a queue-based design was adopted, illustrated in Fig. 6.4. At its core, the *Machine Communication* module interfaces with the machine via Modbus, generating real-time snapshots of operational data, which are placed into the `machine_data` queue.

The *Data Collection* module filters these messages, forwarding relevant snapshots to the `data_to_process` queue. Using Redis as the message broker provides high-throughput, low-latency queuing. The *Data Processing* module parses and validates the snapshots, buffering them until a 60 s observation window is complete. Each snapshot is also logged for traceability and potential model retraining. Completed windows are forwarded to the *Data Classification* module through the `data_to_classify` queue. The embedded ML model outputs a prediction, which is translated into a high-priority com-

Table 6.7: Average response time for the liter classification in both the client-server and queue architectures.

System Type	Client-Server (s)	Queue (s)
.pt model	1.5124	0.1176

mand placed in the `machine_commands` queue. These commands are executed by the *Machine Communication* module. This asynchronous, modular design reduces dependencies, minimizes overhead, and improves fault tolerance. Both the ML model and the queue system run on-device. To guarantee automation, scalability, and continuous improvement, an Machine Learning Operations pipeline was implemented using MLflow [3], hosted on a central server. The *MLflow Model Registry* manages versioning through three stages: *development*, *staging*, and *production*. Models are stored in MinIO[4] Updates are distributed via a scheduled Cron job that retrieves the latest approved model from the central repository. Each Raspberry Pi acts as an independent authenticated node, verifying model integrity via hash checks. Version control and automated rollback ensure uninterrupted operation in case of errors, enabling resilient large-scale deployment.

To assess the suitability of the proposed solution for edge deployment, we evaluated three metrics on a Raspberry Pi 4 under normal operating conditions: execution time, RAM usage, and CPU consumption. All values were logged to CSV files for analysis. The queue-based architecture was compared against a legacy client-server system, with both tested using the model in its native PyTorch `.pt` format Execution Time results (Table 6.7) show that the queue-based system reduced inference latency by 92% compared to the client-server architecture. RAM Usage was measured by subtracting the Raspberry Pi's idle memory from peak usage during inference. The client-server system peaked at 500 MB, while the queue-based system required only 250 MB, achieving a 50% reduction, a critical advantage in memory-constrained edge environments. CPU

[3] github.com/mlflow/mlflow
[4] github.com/minio/minio

Consumption differences were minor: the client-server averaged 8%, while the queue-based system averaged 10%. This slight increase is outweighed by the substantial gains in speed and memory efficiency. Overall, the results confirm that the queue-based system offers superior performance and scalability for edge deployment. Its asynchronous design, reduced memory footprint, and faster inference times make it the most effective solution for real-time industrial applications.

6.6 Chapter Summary

This chapter examined Type 2 In-Field Environments as a challenging High-Stakes domain for deploying Machine Learning. We explored the Carpigiani case study for their gelato and ice cream making machines.

We developed an automated system for correcting human errors, capable of detecting unbalanced mixtures during production and taking corrective action in real-time. This required models that could process multivariate time series data of varying lengths, deliver low-latency responses, and remain efficient enough for deployment on embedded edge devices. To achieve this, we developed the HoT-AI system, centered on a Multi-Milestone Classifier that evaluates mixture balance at four key stages of the production process. Building this system required careful dataset collection and preprocessing, including the creation of multiple fixed-length datasets. We then explored several model architectures, ultimately selecting a CNN and extending it to multiple milestones through a custom transfer learning strategy. Hyperparameter optimization further ensured that the final models achieved real-time inference performance while remaining efficient and lightweight. Finally, the HoT-AI system operationalized edge intelligence through an optimized, queue-based inference mechanism, and the ML model was implemented on the ice cream machine. With a design that is adaptable and transferable to other domains, this deployment offered a solution for real-time error correction in industrial machines.

This concludes the exploration of the Second High-Stakes Environments, demonstrating how In-Field Environments present unique challenges in terms of connectivity, bandwidth, and hardware limitations, yet also provide opportunities for deploying lightweight and efficient edge-first AI solutions. The insights and results obtained here set the stage for addressing progressively more complex and less controlled High-Stakes Environments in the following chapters, where even greater variability, uncertainty, and autonomy requirements will need to be considered.

Chapter 7

Type 3: Disaster-Impacted Environments

Disaster-impacted environments constitute the most extreme and least predictable category within the hierarchy of High-Stakes Environments. In contrast to industrial or in-field deployments (Types 1 and 2), where communication infrastructures may experience limitations but remain largely functional, Type 3 scenarios unfold under conditions where such infrastructures have been severely compromised or entirely destroyed. These environments emerge in the aftermath of natural disasters, armed conflict, or large-scale system failures, where the absence of reliable connectivity and stability poses unprecedented challenges for the intelligent systems [1].

[1] This chapter is based primarily on the following authors' publications:

- S. Dahdal, S. Cavicchi, A. Gilli, et al., "Roamml distributed continual learning: Adaptive and flexible data-driven response for disaster recovery operations," Journal of Network and Computer Applications, p. 104 322, 2025.
- S. Dahdal, F. Poltronieri, et al., "A data mesh approach for enabling data-centric applications at the tactical edge," in 2023 International Conference on Military Communications and Information Systems (ICMCIS), 2023, pp. 1–9
- S. Dahdal, F. Poltronieri, A. Gilli, et al., "Roamml: Distributed machine learning at the tactical

7.1 Rethinking Distributed Training

Machine Learning in disaster-impacted environments must go beyond traditional assumptions of reliable connectivity and centralized control. Standard training pipelines rely on centralized data collection, where raw datasets are transmitted to a central server for consolidation. In DDIL scenarios, this paradigm collapses: transmitting large data volumes is infeasible due to fragile links, high latency, and limited bandwidth, all of which may be further compromised by damaged infrastructure and degraded communication channels [9], [82], [83].

While federated and distributed approaches have been proposed as alternatives, they too carry inherent limitations when deployed under conditions of prolonged disruption. Federated Learning, for instance, presupposes the existence of a central aggregator and stable rounds of communication, assumptions that rarely hold during disaster recovery. Similarly, transfer learning pipelines often require consistent weight synchronization across nodes, a process that cannot be guaranteed when devices experience extended isolation. Even decentralized approaches such as gossip or peer-to-peer learning, though robust to coordinator failures, introduce heavy communication overhead that can overwhelm already constrained links.

The critical distinction of type 3 environments lies in the necessity for training to occur locally and autonomously, with only opportunistic or delay-tolerant synchronization. Unlike type 2, where edge inference is sufficient, disaster-impacted conditions demand that models evolve on-site to capture rapidly shifting data distributions, whether

edge," in MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM), 2023, pp. 33–38.

- S. Dahdal and M. Tortonesi, "Enabling big data and machine learning applications in high-stakes environments," in NOMS 2024-2024 IEEE Network Operations and Management Symposium, IEEE, May 2024, pp. 1–4.
- S. Dahdal, A. Gilli, et al., "Roamml platform: Enabling distributed continual learning for disaster relief operations," in 2024 IEEE Symposium on Computers and Communications (ISCC), 2024, pp. 1–6.

from changing sensor availability, environmental volatility, or emergent mission requirements. This reorientation toward autonomy requires rethinking distributed training strategies to minimize communication, tolerate partitioned operation, and accept that models may remain partially misaligned for extended periods.

To highlight the trade-offs, Table 7.1 compares the main distributed ML techniques across dimensions of advantage, limitation, and suitability for Humanitarian Assistance and Disaster Relief (HADR). The table emphasizes that no single method is sufficient: Federated Learning excels when connectivity is predictable, gossip-based approaches thrive in fully decentralized ad-hoc networks, DiLoCo leverages local computation in low-bandwidth "islands," and centralized continual learning can consolidate insights post-mission. However, each exhibits critical weaknesses when deployed in austere environments, underscoring the need for novel hybrid approaches that combine local autonomy with opportunistic synchronization. Therefore, there is a pressing need for innovative solutions to address these challenges that are resilient, flexible, and adaptable in various HADR scenarios, and they must minimize the use of bandwidth. Therefore, there is a pressing need to develop Distributed Learning solutions specifically designed to effectively handle these challenges in critical situations during disasters and their aftermath.

In edge environments CL approaches face challenges similar to those seen in FL. Because CL often relies on centralized data transfer, it requires data to be sent back to a central server for processing. In edge scenarios, this dependency on centralization introduces delays as data waits to be transmitted to the central location, potentially slowing down the model's ability to update and refine itself quickly. In edge settings, where rapid adaptability is crucial, this latency can impact the model's effectiveness, highlighting the need for decentralized or more locally adaptable CL methods.

Disaster environments are inherently dynamic and resource-constrained, making the allocation of limited communication capacity a complex and often suboptimal process.

These conditions underscore the need for resilient, adaptive systems that can operate under uncertainty and maintain effectiveness despite the collapse of conventional infrastructures. RoamML addresses this gap by proposing a solution that leverages CL to adapt and learn as it iteratively traverses network nodes.

Table 7.1: Comparison of Distributed ML Techniques in HADR Scenarios

Technique	Key Advantages	Key Limitations	HADR Suitability
Federated Learning	• Raw data remain local • Mature ecosystem • Privacy Preservation	• Requires a reachable aggregation server (single point of failure) • Sensitive to intermittent links and stragglers • Edge heterogeneity slows convergence	✓ Best for tactical networks with predictable connectivity × Performance degrades under extended partitions or mobility
Gossip/P2P Learning	• Fully decentralized; • No coordinator needed • Robust to node churn • Small, frequent parameter exchanges	• Slower convergence and higher model variance • Difficult to certify global model correctness • Significant network overhead due to peer-to-peer traffic and frequent exchange of parameters	✓ Effective in disrupted, ad-hoc mesh networks with no central coordination × Communication overhead can burden low-bandwidth or energy-constrained environments
DiLoCo	• Maximizes local compute with minimal communication • Designed for low bandwidth "islands" • Few communication rounds needed	• Assumes homogeneous hardware • Synchronization delays with slow nodes	✓ Well-suited for islanded clusters with periodic backhaul × Performance drops with heterogeneous or mobile devices
Centralized Continual Learning	• Enables continual learning over time • Supports rapid integration of new tasks • Well-suited for dynamic data scenarios	• Requires reliable uplink to central server • Raises privacy concerns due to raw data sharing • Incurs latency in data transfer	✓ Works well when post-mission consolidation is possible × Fails to adapt during disconnected or mission-critical stages

7.2 RoamML Framework: Design and Development

The RoamML framework, short for "Roaming Machine Learning", first presented in [84]–[86], is a novel Distributed Continual Learning approach to train Machine Learning (ML) Models in disaster scenarios. The framework is grounded in the principle that "moving an ML model is more efficient and robust than large dataset transfers or frequent model parameter updates". Therefore, our model will navigate through various data access points of a network to train a model in a Continual Learning and distributed fashion. RoamML acts like a traveler searching for new data sources, thereby continually refining its knowledge base. In this paradigm, the model itself navigates across various data access points in a network, progressively refining its knowledge base through continual and distributed learning. RoamML operates as a "traveler", dynamically seeking new data sources to improve its performance while remaining adaptive to the evolving conditions of disaster scenarios. The framework is specifically tailored for post-disaster contexts, where intermittent connectivity, bandwidth scarcity, and damaged infrastructure impede the use of both conventional and non-traditional ML training strategies. To overcome these challenges, RoamML incorporates dynamic re-routing capabilities. When a connection to a particular node is lost, the framework can immediately assess the data already acquired, select the next optimal node, and re-route the model accordingly. This on-the-fly decision-making ensures that the learning process continues uninterrupted, even under unstable and resource-constrained conditions.

7.2.1 RoamML Architectural Design

A visual illustration of the RoamML framework is provided in Fig.7.1. Conceptually, the framework can be decomposed into two interdependent components: the *Machine Learning part* and the *Routing part*.

The ML part is responsible for the continual training of the ML model. To achieve

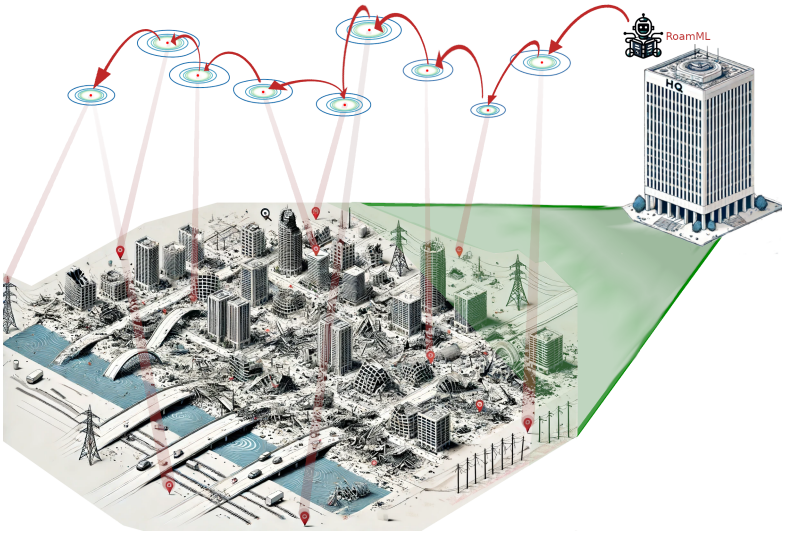


Figure 7.1: RoamML creates situational awareness in HADR scenarios by roaming between network nodes.

robustness under disaster conditions, RoamML employs continual learning algorithms that enable incremental updates across different nodes in a sequential manner. Among the available methods, Experience Replay (ER) was selected due to its effectiveness in mitigating catastrophic forgetting, as demonstrated in our empirical comparisons with alternative continual learning strategies. The ER approach uses a replay buffer to store and retrieve past examples, ensuring that knowledge from earlier tasks is not lost. To limit communication overhead, the buffer size is fixed, thereby constraining its weight contribution during model transfers between nodes. Initially, RoamML followed the standard random replacement strategy of the original ER algorithm. More recently, enhancements inspired by Bugzzega et al. [87] have been introduced, notably the use of a Balanced Reservoir. This modification replaces samples from overrepresented classes with those from minority classes, thus maintaining balance in the buffer. The result is improved model generalization, mitigation of class imbalance, and greater stability during sequential training.

The routing component is tasked with selecting the next node for the RoamML agent,

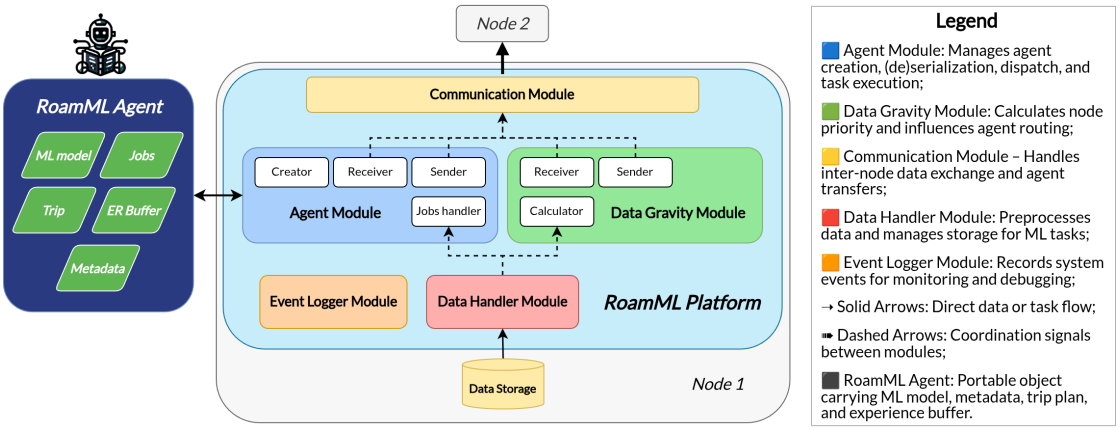


Figure 7.2: The RoamML Platform.

which carries the ML model. Its design is inspired by the concept of Data Gravity (DG), first introduced by McCrory in 2010 [88], which posits that large data volumes exert a gravitational pull, attracting applications, services, and further data. Within RoamML, each unvisited node in the network is modeled as exerting an "attractive force", the intensity of which reflects the node's priority and data importance. Nodes with stronger attraction are more likely to be selected as the next training location. If no suitable unvisited nodes are currently available, the agent waits and periodically searches the network state until a new eligible node is discovered. Once the model migrates to a node and is trained on its data, that node's data gravity is reduced to zero, signifying that its critical information has been assimilated. The node then shifts to a supporting role, functioning as a potential relay or stepping stone to facilitate the model's migration toward other high-priority nodes.

7.2.2 RoamML Platform

In [85], we presented an end-to-end open-source implementation of the RoamML framework (repository available in Section 7.5.6). The architectural design of this platform is illustrated in Fig. 7.2. The RoamML Platform is a modular, extensible architecture de-

signed to integrate seamlessly with edge devices and serve as a landing zone for the RoamML Agent.

The RoamML Agent is a serializable object that encapsulates all the information required for the model's mobility and training across a network. It carries the ML model, which is incrementally updated as the agent visits different nodes, along with computational jobs that specify tasks such as preprocessing and training. In addition, the agent maintains a trip record to track visited and pending nodes, metadata describing its state and context, and an experience replay buffer that stores representative samples to support continual learning.

The RoamML platform functions as the operational core, orchestrating the agent's activities, coordinating training, and managing communication across the network. It is composed of five key modules: The **Communication Module** manages all data exchanges within the platform, supporting multiple protocols including TCP and UDP. This flexibility ensures reliable and adaptable communication for agent transfers in highly variable network conditions. The **Agent Module** handles the lifecycle of the agent, including initialization, serialization for transmission, deserialization upon arrival, and dispatch to the next node. A key sub-component, the Jobs Handler, supervises ML-related tasks such as training and evaluation, ensuring that computation at each node is executed efficiently and correctly.. The **Data Gravity Module** implements the concept of DG by calculating a gravitational pull metric for each node, which reflects the relevance of its data. This score influences routing decisions and updates the agent's trip record in coordination with the Agent Module. A higher gravitational pull increases the likelihood that a node will be selected as the agent's next destination. It is central to the development of the model is the strategy employed by the Agent in determining the optimal path. The **Data Handler Module** provides essential tools for managing data pipelines. It supports preprocessing, data augmentation, and storage operations, ensuring that each node's data is prepared and optimized for effective ML training and evaluation. The **Event Logger**

Module records all system events in a structured format. This capability is essential for debugging, monitoring system performance, and tracking the evolution of the ML model throughout its journey.

In the developed platform, DG requests are computed and processed at each hop, enabling routing decisions to be updated incrementally. This ensures that the agent's trajectory remains responsive to real-time conditions rather than relying on static pre-computed paths. The dynamic nature of this process is crucial for deployment in disaster-impacted and high-stakes environments, where adaptive and resilient decision-making is required for effective model evolution.

7.3 Data Gravity: Choosing the Optimal Path

The concept of Data Gravity is central to the RoamML framework, serving as the heuristic for determining the next hop in the agent's journey. Selecting the optimal node is therefore a critical decision, as it directly influences training efficiency, model robustness, and overall mission success. Early implementations of RoamML employed a simplistic navigation criterion based mainly on dataset size. While effective in small-scale tests, this naive approach quickly proved inadequate for the complexities of HADR scenarios, where data sources are highly heterogeneous and system resources are constrained.

To address these shortcomings, we extended the concept of DG into a multi-dimensional model that incorporates a broader range of factors. Beyond dataset volume, the model now evaluates hardware capabilities (e.g., processing power, memory, and energy availability), software readiness (e.g., compatibility and framework support), and intrinsic data properties (e.g., relevance, diversity, and quality). It also accounts for scenario-specific aspects such as network stability, urgency of the mission, and potential risks tied to the environment. The challenge lies in simultaneously weighing multiple, often

competing, features, each measured on different scales and units. By framing node selection as a multi-criteria decision-making problem, the expanded DG model allows the agent to balance trade-offs adaptively, leading to more informed and context-aware routing decisions. Ultimately, this enriched approach provides a more effective mechanism for training ML models under disaster conditions, where robustness and adaptability are as important as computational efficiency.

7.3.1 Data Gravity Model

In our earlier work [85], we introduced a baseline model for node selection within a network, where priority was determined solely by the volume of available data (measured in MB/GB). The intuition behind this approach was straightforward: richer datasets provide more learning material, yield deeper insights, and support stronger generalization. While effective as a starting point, this strategy can be considered overly simplistic. A selection mechanism based only on dataset size does not necessarily align with dynamic mission objectives or the operational constraints of disaster environments. For instance, a node with abundant data may be located in a region with limited computational resources, unstable power supply, or poor connectivity, thereby hindering its actual utility. To overcome these limitations, a more robust framework was developed referred to as the RoamML Data Gravity Model. This enhanced model extends the concept of data-centric selection by incorporating three principal dimensions: *Dataset Information* and *Location Resource Constraints*, and the *Performance of the ML model* being trained by the Roaming Agent. By jointly considering these aspects, the RoamML Data Gravity Model transforms node selection into a Multi-Criteria Decision Making process. This enables the roaming agent to prioritize nodes not merely by the volume of data they hold, but by their holistic contribution to model improvement under the prevailing operational conditions. A detailed summary of these variables and their roles in influencing the node-selection process is presented in Table 7.2.

Dataset Related Information

When refining node selection strategies, it is imperative to consider multiple aspects of the datasets available at each node, beyond simply the *volume* of data. Key factors such as *data distribution* - including variability and the presence of out-of-distribution samples - must be taken into account, as selecting nodes that provide diverse and representative data is essential for reducing model bias. The Shannon Entropy Diversity Index (H) quantifies the randomness, evenness, and richness of a dataset, making it an effective metric for assessing distributional balance. In disaster zones, data sources are often fragmented (e.g., partial sensor coverage, uneven UAV deployments). High entropy ensures the agent prioritizes nodes with diverse, representative samples, mitigating bias in models trained on skewed or incomplete data [89]. Equally important is evaluating the *quality* of the data [90], [91]. This could be achieved qualitatively by assessing, on the one hand, the completeness and consistency of the information. For instance, a node collecting wildfire sensor data might lack critical air quality measurements due to damaged equipment, reducing its utility for health risk modeling. Also, discrepancies in timestamp formats (e.g., UTC vs. local time) across UAV nodes could misalign event logs, delaying rescue coordination. On the other hand, it is essential to consider the alignment between the data stored at each node and the Agent's operational objectives.. For example, during a flood response, nodes storing historical traffic patterns may be de-prioritized over those providing real-time hydrological data, which directly supports evacuation route planning.

In addition, disaster edge scenarios are characterized by a continuous flux of information, driven by the inherent instability of such conditions. Hence, the *freshness* of data becomes a pivotal parameter. It is typically measured in terms of how recent the information is (Age), ensuring that the model is constantly exposed to the latest data, which is vital for real-time decision-making. Prioritizing up-to-date information supports the model in adapting to evolving conditions and enhances its effectiveness in critical situa-

tions. In general, the value of data and information is essential to be considered [92] as it plays a significant role in such environments. It is not only the timeliness of the data that matters, but also its relevance and impact on decision-making processes.

Location Resource Constraints

It is also crucial to consider the hardware and software features of the infield located nodes. Among these, the CPU is a critical component that must be carefully evaluated to ensure it can meet the demands of complex model training and data processing. The *CPU power* - in terms of computational capability - is primarily influenced by its frequency (GHz) and number of cores, which together determine its speed and ability to handle parallel tasks. High-performance CPUs can efficiently manage intensive operations, thereby accelerating training iterations and reducing overall processing time. Additionally, monitoring the mean values of both *CPU load percentage* (Util_{CPU}) provides insights into its operational health, efficiency, and level of computational demand. Intensive CPU usage for extended periods can strain the system, increase power consumption, and generate excessive heat, leading to performance degradation and potential hardware failures. A low level of CPU efficiency could profoundly impede accurate model training, especially as it requires multiple taxing operations [93], [94]. Nonetheless, the presence of a *GPU* within a node greatly boosts its computational abilities, essential for efficiently managing the rigorous demands of machine learning training and large data processing. These valuable advantages make the GPU an instrument of utmost importance, emphasizing the need for a meticulous evaluation of its properties in order to select the best performing one. Key considerations for a *GPU* include its *architecture*, typically expressed in nanometers (nm), as it directly influences its energy and thermal efficiency, as well as overall performance. Equally important are the GPU's *utilization percentage* (Util_{GPU}) and computational *power*, where all considerations previously outlined for the CPU's properties remain relevant [93]–[95]. For instance, Computational capacity di-

rectly impacts mission-critical timelines; therefore, nodes with multi-core CPUs (e.g., 8-core @ 3.5 GHz) and GPUs (e.g., 7nm architecture, $>$ 20 TFLOPS) accelerate model updates during rapid situational changes, such as evolving flood zones or wildfire spread [96]. Measuring the *total* and *available memory* (RAM), in gigabytes and monitoring the percentage of *storage utilization* ($\text{Util}_{storage}$) could favor a node capable of ensuring efficient data storage and retrieval over one that may struggle to accommodate the dataset or model parameters, potentially leading to degraded performance or training failures. Energy efficiency (e.g., CPU power $< 50W$) is equally important, as nodes often operate on limited battery reserves. This is particularly critical in disaster-affected environments, where access to external power sources may be restricted or unavailable. In such cases, monitoring a node's *power consumption* (W), *remaining battery life* (in hours), and *battery capacity* (mAh) becomes essential to ensure reliable operation and to prevent unexpected shutdowns during active learning sessions, failures that could severely disrupt the overall training process. Nodes particularly low on battery may not be suitable to sustain heavy computational workloads, as they would risk premature power-off. This underscores the advantages of adaptive node selection strategies, which can dynamically adjust the Agent's path based on real-time conditions, maximizing the utilization of available resources and optimizing the training progress [93], [94].

Additionally, failing to address issues such as slow and inefficient data transfer, poor performance, and delayed response times could drastically impede successful model training and swift movement. To accomplish this, a first step involves favoring low-latency network connections, as they allow timely communication between nodes, rapid data and Agent exchange. *Latency*, which can be measured in milliseconds (ms), is critical for assessing network responsiveness and enabling a seamless progression of the learning process [97]. Similarly, another critical feature to evaluate is the *bandwidth utilization*, typically expressed in megabits per second (Mbps), as it reflects the workload of the entire infrastructure. The greater the bandwidth, the larger the volumes of infor-

mation and data that can be efficiently transferred [95]. Conversely, low transmission capacity results in bottlenecks and congestion, slowing down the Agent's movement and delaying the convergence of the model. For instance, Concurrent data streams (sensor alerts, GPS trajectories, weather forecasts) require prioritized bandwidth allocation to avoid congestion during mass evacuations. Equally fundamental is monitoring *throughput*, generally measured in Mbps, reflecting the network's overall capacity to handle data traffic efficiently. Maximizing this metric leads to smooth and rapid information transfer, crucial for improving the performance of distributed ML tasks. To further refine the path-determination strategy, the outlined features need to be integrated with other context-specific aspects. Specifically, in Human Assistance & Disaster Recovery scenarios, where non-stationary data sources like UAV are present, incorporating mobility-aware strategies into the decision-making process is crucial to enable dynamic data acquisition and real-time adaptation.

Termination of the Roaming

Optimizing the movement of the RoamML Agent also requires defining clear termination conditions for its roaming process. Determining when the agent should conclude its journey is critical to balancing model performance with resource efficiency. A central factor in this decision is the performance trajectory of the ML model. Accuracy metrics often serve as the primary benchmark, acting as indicators of the model's readiness for deployment. By establishing a performance threshold, the Agent can autonomously decide to halt further roaming and return to the base station once this level of accuracy is reached. This strategy ensures that training efforts are concentrated on high-impact data sources rather than expending resources on marginal improvements.

Complementary to accuracy, loss functions provide a finer-grained perspective on learning progress. Monitoring measures such as mean squared error for regression, cross-entropy for classification, or Intersection over Union for detection tasks enable

Table 7.2: Possible Node Selection Variables of the Data Gravity Model

Category	Variable	Description	Metric/Formula
Dataset	Volume	Amount of data	size in MB/GB
	Distribution	Data evenness	Entropy $H = -\sum p_i \log p_i$
	Quality	Completeness	Audit score (0–1 scale)
	Freshness	Recency of data	Age $= t_{\text{current}} - t_{\text{update}}$
Hardware	CPU Power	Processing capability	num. cores & clock speed (GHz)
	CPU Load	Processor usage level	$\text{Util}_{\text{CPU}} = \frac{\text{Used}}{\text{Total}} \times 100\%$
	CPU temperature	Processor heat level	°C
	GPU Architecture	Graphics processor size	nm
	GPU Load	Graphics processor usage	$\text{Util}_{\text{GPU}} = \frac{\text{Used}}{\text{Total}} \times 100\%$
	GPU Power	Parallel compute power	TFLOPS, VRAM bandwidth
	Total RAM	Volatile working memory	GB
	Available RAM	Free working memory space	MB
	Storage	Persistent memory usage	$\text{Util}_{\text{storage}} = \frac{\text{Used}}{\text{Total}} \times 100\%$
Power	Consumption	Energy use	Watts (real-time)
	Battery Capacity	Charge storage capability	mAh
	Battery Life	Operational duration	$\frac{\text{Capacity (mAh)}}{\text{Draw (mA)}}$ (hours)
Network	Latency	Response delay	in milliseconds or set a threshold
	Bandwidth	Max transmission capacity	$\frac{\text{Max transferrable data (MB)}}{\text{Time (s)}}$
	Throughput	Data transfer rate	$\frac{\text{Data (MB)}}{\text{Time (s)}}$

the system to track not only whether the model is improving, but also how effectively it adapts to new inputs. Convergence of the loss function, or the absence of meaningful improvement over several training cycles, may signal an appropriate termination point. In addition to performance-based criteria, time-based strategies can be employed to align roaming duration with mission constraints. For instance, a fixed time budget or maximum number of iterations ensures that training remains compatible with real-world operational limits, such as energy availability, network dynamics, or mission timelines.

Finally, a natural conclusion occurs when all available nodes have been visited and their data integrated into the model. In this case, the agent returns to its origin, carrying with it a consolidated model that reflects the full scope of the encountered data. Through this combination of accuracy thresholds, loss convergence, time-based constraints, and coverage completion, the RoamML framework provides flexible yet robust criteria for deciding when roaming should end.

7.3.2 MCDM for Choosing the next hop

Since the enhanced DG Model incorporates multiple parameters, the routing process, particularly the selection of the next hop, must be guided by comprehensive and transparent criteria. To achieve this, the RoamML framework employs a *Multi-Criteria Decision Making* algorithm, which systematically evaluates nodes across diverse factors and identifies the most suitable destination for the RoamML Agent. By applying MCDM algorithms, the DG Model combines both subjective and objective perspectives, enabling adaptive and context-sensitive decision-making. *Subjective Criteria Weighting* involves incorporating expert knowledge and operational insights. Disaster response professionals can adjust weights based on mission-specific priorities, such as the urgency of certain datasets, the importance of geographic proximity to critical zones, or the relative priority of different response tasks. These inputs embed domain expertise into the routing logic, making the system sensitive to the evolving demands of disaster management. *Objec-*

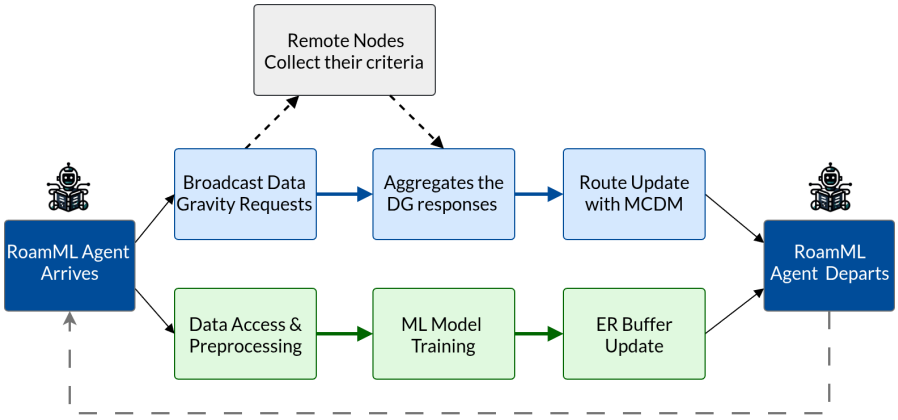


Figure 7.3: The RoamML workflow

tive Criteria Weighting, on the other hand, relies on quantifiable and measurable indicators. Examples include signal strength, node computational capacity, available storage, energy constraints, and dataset volume. These metrics provide an empirical basis for evaluating candidate nodes, ensuring that the agent's routing decisions are anchored in operational feasibility and resource availability.

A key strength of using MCDM lies in its ability to flexibly update weights in real time. For example, during the initial phases of disaster response, network connectivity might be prioritized to stabilize communications. As the situation evolves, decision-makers may shift emphasis toward accessing data from high-priority disaster zones or ensuring model adaptation to rapidly changing conditions. This dynamic reweighting allows the agent to remain agile and responsive under shifting mission constraints. By empowering the RoamML Agent to incorporate both subjective and objective assessments, MCDM ensures that next-hop selection is not only robust but also context-aware. This hybrid decision-making process enhances flexibility, responsiveness, and alignment with the operational realities of disaster management.

Finally, Fig. 7.3 illustrates the overall workflow of the RoamML framework, emphasizing the agent's decision-making loop: integrating DG with MCDM for dynamic routing, while simultaneously training the ML model using the designated CL strategy.

7.4 RoamML Testing Methodology

Evaluating the RoamML framework requires carefully defining experimental conditions to replicate realistic operating environments. The testing methodology involves three key design choices: dataset selection, network emulation, and system configuration.

7.4.1 Testing Configuration

The first step involves distributing a supervised Machine Learning dataset across different nodes. This setup mimics disaster-response environments, where data is inherently fragmented and not centrally available. A variety of datasets can be employed depending on the task: Various datasets can be used, such as MNIST, CIFAR10, mini-ImageNet, and ImageNet for classification tasks. For object detection tasks, datasets like Common Objects in Context (COCO) and Pascal VOC (Visual Object Classes) can be utilized. In principle, any supervised ML dataset can be used within the RoamML framework. For this work, we focused on supervised classification tasks, as they provide a controlled and well-understood setting for evaluating distributed continual learning.

The second step is the selection of emulation software to replicate real-world networking conditions, including intermittent connectivity and node mobility. Tools such as EMANE (Extendable Mobile Ad-hoc Network Emulator) or Mininet-WiFi are particularly suited for this purpose, as they allow researchers to simulate wireless communication, ad-hoc routing, and mobility patterns under controlled yet realistic conditions. These simulations ensure that RoamML is tested against challenges consistent with Disrupted, Degraded, Intermittent, and Low-Bandwidth network environments. Once the dataset and emulation environment are defined, the RoamML Platform and Agent are configured. The implementation is open-source and publicly available (see Section 7.5.6), providing a flexible and reproducible foundation for experimentation. This configuration includes defining the size and replacement strategy of the ER buffer,

which governs how previous experiences are stored and reused for continual learning. At the same time, the neural network architecture must be specified, with parameters such as depth, width, activation functions, and learning rate tailored to the needs of the task. These design decisions directly influence how effectively the framework supports incremental training as the Agent traverses the network.

7.4.2 RoamML Performance Metrics

The evaluation of RoamML requires a set of performance metrics that reflect both the efficiency of its operation and the effectiveness of the resulting ML model. In this work, we focus on three representative dimensions: communication overhead, experiment duration, and model performance.

Bandwidth consumption serves as the primary indicator of communication efficiency. This can be quantified by monitoring and aggregating the total volume of packets transmitted during an experiment using packet-capturing tools. Since bandwidth is one of the most critical constraints in disaster-response environments, minimizing communication overhead is essential for validating the practicality of the framework. Experiment duration provides a measure of operational timeliness. It is determined by recording the start time at the launch of the agent and the end time when the trained model returns to the base node. This metric reflects how quickly the system can complete a roaming cycle, which is directly relevant to mission-critical contexts where decisions must be made under strict temporal constraints. Model performance represents the quality of the learned representation. This is assessed using established metrics from the Continual Learning community. Accuracy provides a straightforward measure of predictive performance, while forgetting metrics evaluate how well the model retains previously acquired knowledge as it encounters new tasks. In scenarios where the test set is balanced across classes, these metrics together provide a reliable picture of how effectively RoamML supports continual adaptation without catastrophic forgetting.

Since RoamML is aimed to be used in real use cases, we follow the description by Van de Ven et al. in [46]. The authors distinguished clearly between the "context set", which represents the underlying distributions of individual tasks, and the "data stream", which is a sequence of experiences presented to the algorithm over time. Each experience includes a set of observations drawn from one or more of these context sets. Real-world Continual Learning algorithms would face many experiences, and they would consist of a sample of the training data of multiple contexts (tasks, classes, etc). Following this definition, we have adapted the accuracy and forgetting metrics in Continual Learning literature, which are clearly presented in [98] for real-world RoamML scenarios. Assuming a complete test set consisting of various tasks of different contexts, we define the Accuracy - average classification Accuracy computed over all classes - at the k^{th} visited node as:

$$acc_k^{test} = \frac{Correct\ Predictions}{All\ Predictions} \tag{7.1}$$

The accuracy metric is selected because it directly quantifies the model's generalization capability across all previously encountered contexts and, to be more precise, the Average accuracy over all classes. The *complete test set* ensures fairness by evaluating performance on all tasks, avoiding bias toward recent contexts. It is particularly useful for deployment-phase validation.

Secondly, the forgetting metric is defined as the difference between the maximum knowledge gained throughout the learning process in the past and the knowledge the model currently has. In other words, this metric estimates how much the model has forgotten given its current conditions. We present an adapted formula from the one presented by Chaudhry et al. in [98] on the complete dataset up to the k^{th} node visited by RoamML as:

$$Forget_k = \max_{l \in \{1,..,k-1\}} acc_l^{test} - acc_k^{test} \ , \ \forall l < k \tag{7.2}$$

The $Forget_k$ metric isolates catastrophic forgetting by comparing current performance (acc_k^{test}) to the peak historical accuracy ($\max acc_l^{test}$). This is critical for diagnosing *stability-plasticity trade-offs*, as positive values indicate knowledge loss. The calculation ensures localized forgetting events are not obscured by averaging. Note, positive $Forget_k$ implies the quantity of maximum forgetting happened until node k^{th} node. A negative forgetting means that the model has gained knowledge since the previous evaluations.

Finally, we have adapted the *Average Forgetting* which can be another way to summarize how much of the average forgetting until the k^{th} node, which can be adapted as:

$$Average\_Forgetting = \frac{1}{k-1} \sum_{j=2}^{k} Forget_j \tag{7.3}$$

The $Average\_Forgetting$ metric aggregates forgetting across all nodes to evaluate systemic knowledge retention. By smoothing transient fluctuations, it highlights long-term trends, making it ideal for comparing other lifelong learning strategies in the future. The $\frac{1}{k-1}$ normalization ensures scalability across varying journey lengths. The Average Forgetting metrics will be of great help in comparing the various choices and configurations of the RoamML framework. This metric is especially valuable because it provides a comprehensive and summarized measure of the model's ability to retain knowledge throughout the Agent's journey. By capturing the cumulative impact of forgetting across all visited nodes, the Average Forgetting metric allows us to assess the model's consistency in retaining past knowledge while adapting to new data. For clarity, we briefly define the key symbols used in our metrics: k indexes the current node in the agent's journey, while l and j represent past nodes ($1 \leq l, j < k$). The accuracy acc_k^{test} is

computed on the complete test set at node k, and $Forget_k$ measures the knowledge loss (positive values indicate forgetting). The $Average\_Forgetting$ averages these values across all visited nodes.

To further clarify, RoamML is evaluated by computing the model's accuracy (Eq. 7.1) on a test set after each training cycle at every node. This enables us to monitor the model's learning progression as the agent moves across the network. In addition to accuracy, we compute Forgetting (Eq. 7.2), Average Forgetting (Eq. 7.3), and loss metrics to provide a more comprehensive view of performance trends, in alignment with standard Continual Learning literature [98]. While these intermediate evaluations help track learning dynamics throughout the roaming process, we report the final test accuracy, final average forgetting, and final loss as the main performance indicators. These metrics are commonly adopted in the literature for evaluating real-world Continual Learning scenarios [46], [99], as they best reflect the model's overall effectiveness and deployment readiness.

7.5 Experimental Evaluation

To validate the proposed RoamML framework and the enhanced Data Gravity (DG) model, we designed an experimental setup using the CIFAR-10 dataset. The goal of these experiments is to demonstrate the ability of the framework to achieve distributed, network-wide training with competitive accuracy while significantly reducing bandwidth consumption.

The experiments were conducted using the Mininet-WiFi emulator, chosen for its ability to simulate dynamic and realistic wireless network environments. Nodes were distributed across a 250 × 250 square meter area to emulate scaled-down, yet representative, conditions. Within this environment, a total of 15 nodes were deployed, positioned at varying distances such that partial disconnections could occur. Communica-

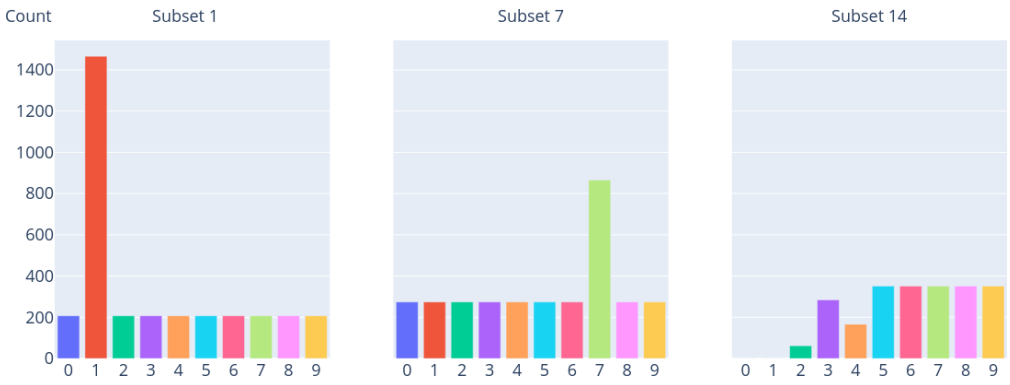


Figure 7.4: Example of distributed data subsets assigned to emulation nodes

tion relied on a peer-to-peer ad hoc network configured with the Optimized Link State Routing Version 2 (OLSRv2) protocol. For the radio propagation model, we adopted the Log-Distance model with a path-loss exponent of 3.5, a parameter widely used for characterizing Wi-Fi signal attenuation in urban scenarios [100].

For the learning task, we selected CIFAR-10 as a benchmark dataset. CIFAR-10 is well established in the supervised classification community and is widely adopted in the literature, making it suitable for reproducibility and comparability of results. The dataset also aligns with Human Assistance & Disaster Recovery operational needs, particularly in the context of object recognition from imagery. CIFAR-10 contains 50,000 training images and 10,000 test images across ten balanced classes, while its size makes it computationally manageable for our experimental framework.

To emulate the challenges of disaster-response settings, the dataset was partitioned into 15 unbalanced subsets, one for each node. As illustrated in Fig. 7.4, this distribution ensures heterogeneity, creating a non-i.i.d. environment where certain nodes contain only minority classes or reduced sample counts. This setting directly challenges the RoamML Agent to maintain global accuracy despite encountering fragmented and unevenly distributed data across the network.

Each node was configured given 8 properties (chosen from the described DG Model presented in Section 7.3) to be assessed upon, therefore emulating a real-world, het-

erogeneous network environment. All the criteria were assigned to each node as a *node_metadata.json* file that would be updated, if needed, at each request of the DG information. The criteria are the following: The *CPU* and *GPU* capabilities were evaluated based on the number of cores multiplied by their frequency. This metric provides insight into the computational power available at each node. *Dataset size* measured in MB, assessing the quantity that each node contains. Also, the *Data Distribution*, in other terms, the balance of the dataset was measured using the Shannon Equitability Index, which ranges from 0 to 1. A fixed *RAM size* in GB, in the range $[4 \rightarrow 16]$ GB, which affects the node's ability to process data. *Battery capacity*, measured in milliampere (mA), is in the range $[1500 \rightarrow 7000]$ mA. *Bandwidth*, quantified in megabits per second (Mbps), evaluates the data transfer capacity of each node, essential for efficient communication within the network. *Latency*, recorded in milliseconds, impacts the responsiveness of the network and the speed at which data can be transferred between nodes. Latency can be approximated by measuring the response times to a data gravity request. Plus, RoamML can extrapolate an approximation of these values by the underlying routing protocol, e.g., OLSRv2.

The criteria and their effects are reported in the upper half of Table 7.4. In all experiments, we opted for a convolutional neural network (CNN) architecture shown in Table 7.3. All configurations of the experiment can be seen in the shared repository (see Section 7.5.6)

Specifically, we conducted various experiments. The first experiment is to define a centralized baseline scenario, which establishes a performance baseline for our ML model and quantifies the hypothesized bandwidth utilization if we had to move the data to a centralized node. The second experiment evaluates the performance of FL in an ideal scenario and defines a baseline for distributed learning methodologies. The remaining three experiments were done via the RoamML framework. The first of these was a *Single-Criteria scenario*, showcasing the original routing strategy via a DG model based solely

Table 7.3: CNN Architecture

Layer(s)	**Configuration**
Input	32x32x3 RGB image
Conv2D + BatchNorm (x2) MaxPooling2D + Dropout	32 filters, 3x3, ReLU 2x2, rate=0.3
Conv2D + BatchNorm (x2) MaxPooling2D + Dropout	64 filters, 3x3, ReLU 2x2, rate=0.5
Conv2D + BatchNorm (x2) MaxPooling2D + Dropout	128 filters, 3x3, ReLU 2x2, rate=0.5
Dense + BatchNorm + Dropout Dense	128 units, ReLU, rate=0.5 10 units, Softmax

on the size of the dataset presented by each node. The remaining two experiments were based on the newly developed DG Model through the multi-criteria decision-making process. The first of the remaining two was called *context-aware scenario*, where we assume an informed domain expert brings specialized knowledge and experience relevant to the specific application domain scenario to set weights for the different criteria. However, in some scenarios, the characteristic of having little to no information about the situations can result in an absence of human judgment, thereby making the weighting of criteria imprecise and, in some cases, impossible; therefore, we define the second scenario called *context-independent scenario*.

7.5.1 Ideal Centralized Baseline Scenario

In the "centralized baseline scenario", the headquarters (HQ) node aggregates data subsets from all infield nodes and then proceeds to train a centralized model. Throughout this emulation, we observed a total bandwidth consumption of approximately **4141.5MB** $\approx$ **4.0GB**. On the other hand, in terms of machine learning performance, the centralized

model demonstrated an **accuracy of 83%** and **0.51 of loss** on the test set, thereby setting a benchmark for the desired performance.

In terms of the total time spent to complete the experiment, we have considered the time required to move the dataset to the centralized node plus the training time, resulting in a total time for transfer of **2008 seconds** $\approx$ **33.5 minutes**. The training time highly depends on the available GPU, as in a real-world scenario, the primary impact would stem from the time taken to transfer all the data to a central location. This is because the model training will likely be performed using a GPU or TPU, but it will start only after the data transfer is complete. In our case, we used an NVIDIA TESLA K80 GPU, and the training time was about **294 seconds** $\approx$ **5 minutes** for 20 epochs. The total time was about **2302 seconds** $\approx$ **38.5 minutes**. Note that this result is based on an ideal and unrealistic assumption that all nodes are able to use their link at full bandwidth to transfer the data to the centralized node. To make a more realistic assumption, let's assume that the nodes can use half of the available bandwidth. Therefore, the time needed to transfer the dataset and then train the ML model would be at least **4300 seconds** $\approx$ **71.6 minutes**.

7.5.2 Ideal Federated Learning Scenario

We established the "distributed baseline" using Federated Learning (FL) to assess the performance of distributed model training across decentralized data sources. Instead of deploying an emulated field network, we conducted simulations using the Flower framework on a cloud-based NVIDIA H100 GPU, which provided the computational capacity to simulate 15 concurrent clients. Each simulated client represented a node with its own distributed subset of the CIFAR-10 dataset. The FL setup employed the Federated Averaging (FedAvg) algorithm over 15 federated rounds, requiring a minimum of **15 available clients** per round to ensure full participation. During each round, all clients independently trained local models on their respective data subsets for 15 epochs. Upon completion of local training, model parameters were aggregated at the central server.

This global model was then evaluated against the centralized CIFAR-10 test set to monitor performance throughout the training process. The final evaluation of the FL-based model yielded an **accuracy of 80%** and a **test loss of 0.72**. These results provide a strong and realistic baseline for distributed learning approaches and serve as a reference point for evaluating more dynamic methods like RoamML, particularly in scenarios where data remains decentralized and network conditions may vary. Moreover, this setup assumes ideal network conditions and does not address the communication and synchronization constraints highlighted in Section 7.1, which are critical in DDIL environments.

7.5.3 RoamML Single-Criteria Scenarios

In the following part of the experiments, we will be testing the ability of RoamML in various cases. Starting with the original single-criteria DG model. In this experiment, the choice of the next node is focused solely on the size of the dataset. Therefore, the size of the dataset has a complete influence on the choice of the next node to hop to. Starting with network usage, the total UDP bandwidth consumption, including both the multicast DG requests and the unicast responses, amounted to **122KB** for the entire scenario, representing a negligible overhead. Additionally, the TCP bandwidth consumption related to the RoamML Agent's roaming activity was approximately **1244MB**. The entire training took approximately **4580s** $\approx$ **76min**.

The performance of the RoamML model in single criteria is presented in Fig. 7.5a, which shows Grey dashed vertical lines to denote the completion of training on specific nodes, as indicated on the x-axis, the time unit on the lower x-axis, and two line to depict the accuracy and the loss. In the same Figure, we can observe on the top x-axis the order of the nodes the RoamML Agent has chosen to travel to, and on the bottom x-axis the absolute time elapsed in seconds. The Green and Violet dashed vertical lines illustrate (in seconds) the needed time for respectively the ideal full bandwidth and the realistic half bandwidth network condition of the baseline scenarios to be completed. We can also

Table 7.4: Experimental Scenario Criteria

Type	Profit							Cost
Criteria	Data Volume	Data Distribution	CPU Power	GPU Power	Bandwidth	Total RAM	Battery Capacity	Latency
Effect	(+)	(+)	(+)	(+)	(+)	(+)	(+)	(−)

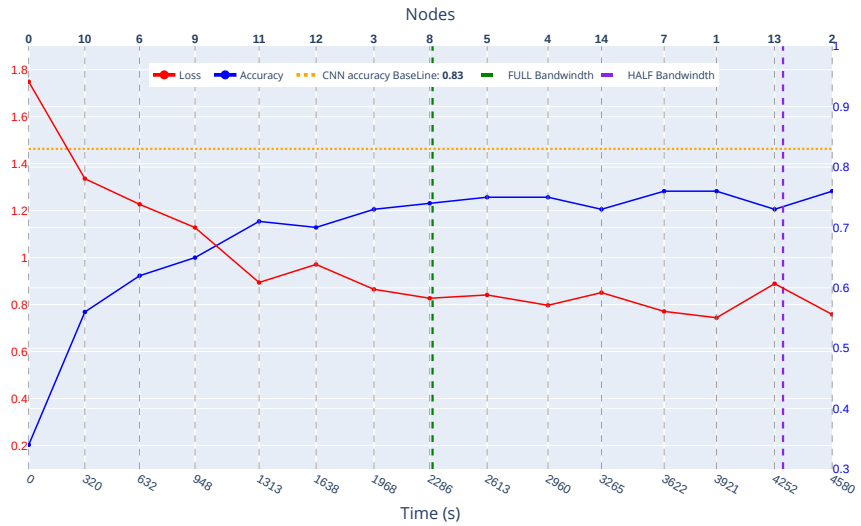
Best Worst Method								
MIC	3	1	3	4	4	6	7	9
LIC	7	9	3	4	4	3	3	1

MIC: Represents the comparisons between the best criterion (Data Distribution) and all other criteria. E.g., the result of the comparison between Data Distribution and Bandwidth is 4. This means that Data Distribution is considered four times more influential than Bandwidth in the decision-making process.
LIC: Represents the comparisons between all criteria and the worst criterion (Latency). E.g., in the LIC list, the result of the comparison between CPU Power and Latency is 3, indicating that CPU Power is considered three times more important than Latency according to the preference scale.

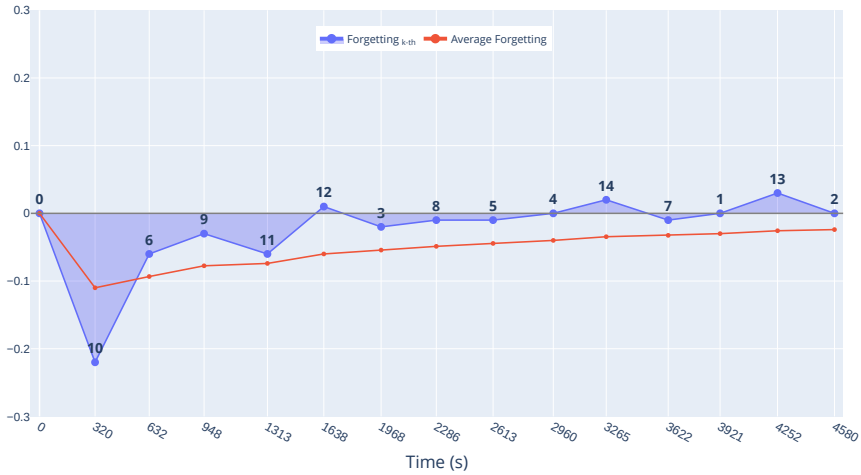
observe a constant decrease in the **test loss of 0.758** after training on each node. On the other hand, we can note a steady increase in the test reaching an **accuracy of 76%**. As in Fig. 7.5b, we can observe the Forgetting Metric and the **Average Forgetting**, which at the end of the trip in this scenario was approximately **-0.0239**.

7.5.4 Multi-Criteria Context-aware scenario

To have comparable experiments, we opted for a unified node alternatives ranking algorithm, that is the *Technique for Order of Preference by Similarity to Ideal Solution (TOPSIS)*. The choice of TOPSIS was primarily due to its basic principle that the optimal alternative should be closest to the ideal positive solution and farthest from the



(a) Accuracy Metric



(b) Forgetting Metric

Figure 7.5: Performance of RoamML under the Single-Criteria Scenario. (a) **Accuracy Metric:** Tracks model accuracy and loss across the sequence of nodes over time. The horizontal yellow line indicates the centralized CNN baseline (83% accuracy). (b) **Forgetting Metric:** Illustrates model's forgetting metrics across training, with individual forgetting values (blue) and the average forgetting trend (red). We can observe two slight drops in performance at nodes 13 and 14, which are attributed to their highly skewed and unbalanced data distributions. The distributional shift increases the difficulty of learning therefore higher forgetting.

negative one. At its core, TOPSIS calculates the geometric distance to determine the relative proximity of each option, then ranks the alternatives based on these distances. Its advantages include simplicity, intuitiveness, and computational efficiency, making it well-suited for handling trade-offs between criteria. This approach has also been empirically validated in numerous use cases [101]. On the other hand, the selection of the weighting criteria varied between experiments.

Therefore, the domain expert assigns weights to the criteria using the *Best Worst Method (BWM)*. This method requires, as input, the creation of two lists of values: the *Most Important Criteria vs. other criteria (MIC)* and the *Least Important Criteria vs. other criteria (LIC)*. MIC records the comparisons between the best criterion (Data Distribution in our case, as shown in Table 7.4), and all other criteria. Where a higher value indicates a greater relative dominance of the most important attribute over the others. Conversely, LIC reflects the preferences of all criteria over the least influential one (Latency in our case, as shown in Table 7.4). For each comparison, a lower value indicates less relative dominance of the attribute compared to the worst criterion over the latter. It is important to note that the selection of the most and least important attributes in the decision-making process is subjective and can be adjusted as needed. This is a proven and robust methodology that is compatible with the high-stakes and urgent nature of decision making in HADR. It therefore provides a more context-aware scenario for the RoamML Agent to select the next node to train on.

Starting with network metrics results, the total UDP bandwidth consumption, including both the multicast DG requests and the unicast responses, reached **94KB** representing a negligible overhead. Additionally, the TCP bandwidth consumption related to the RoamML Agent's roaming activity was approximately **789MB**. The entire training took approximately **4482s** $\approx$ **74.5min**. The performance of the RoamML model in context-aware is presented in Fig. 7.6a, which shows grey dashed vertical lines to denote the completion of training on specific nodes, as indicated on the upper x-axis, the time unit

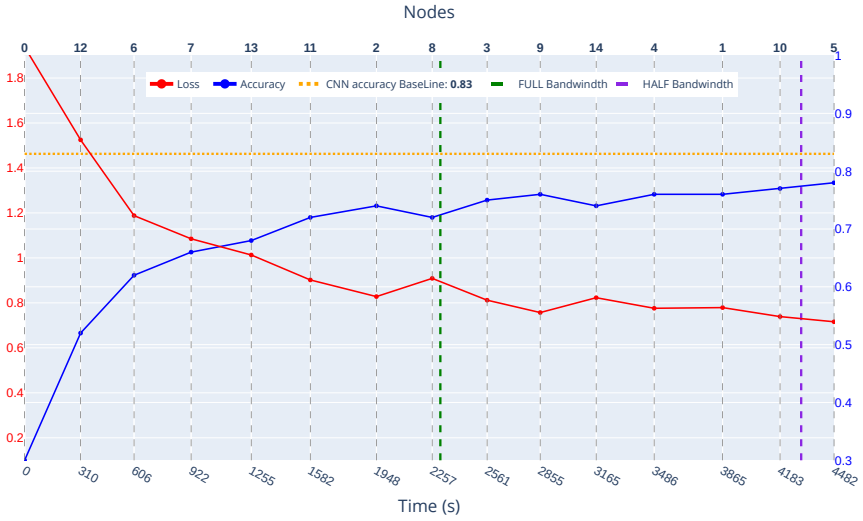
on the lower x-axis, and two lines to depict the accuracy and the loss. In the same Figure, we can observe on the x-axis the order of nodes that the RoamML Agent has chosen to travel to. This ordering illustrates the strategic decisions made by the Agent in prioritizing node visits via MCDM based on the combo of BWM and TOPSIS *BWM-TOPSIS*, which has shown significant results in [102] on the test set. We can also observe a constant decrease in the **test loss of 0.716** after each node. On the other hand, we can note a steady increase in the test reaching an **accuracy of 78%**. As in in Fig. 7.6b we can observe the Forgetting Metrics and the **Average Forgetting** computed at the end of the trip was approximately **-0.0293**.

7.5.5 Multi-Criteria Context-independent scenario

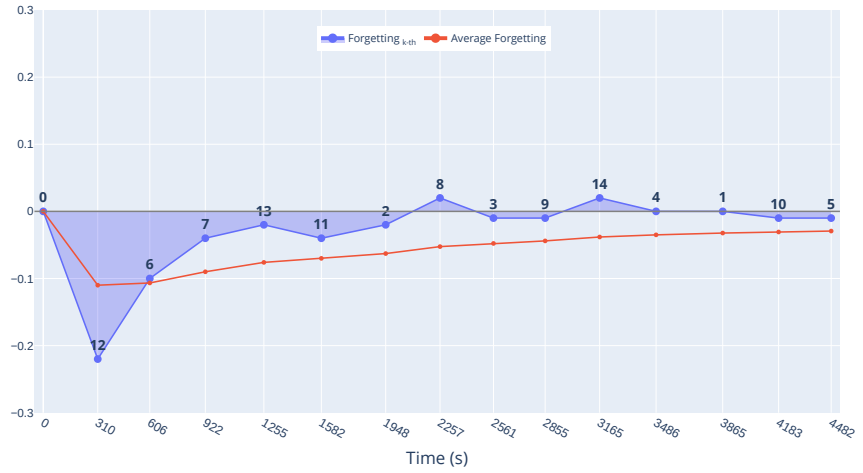
When it is not possible to weigh the criteria due to a lack of contextual awareness, alternative strategies must be adopted. To address this, we designed a second experiment in which the context-aware approach is deliberately excluded - a setting we refer to as context-independent. Accordingly, we employed a more objective, context-agnostic strategy by adopting the *Entropy weighting method*. This method, which draws from probability theory, associates the relevance of each criterion with its level of uncertainty, offering the RoamML Agent a fully data-driven approach that does not rely on human feedback. This consideration ensures that the weighting process remains unbiased and consistent across different scenarios, thereby enhancing the robustness of the decision-making framework.

On the networking level, the total UDP bandwidth consumption amounted to **109KB** for the entire scenario, representing also a negligible overhead. Additionally, the TCP bandwidth consumption related to the RoamML Agent's roaming activity was approximately **931MB**. The entire training took approximately **4451s** $\approx$ **74min**

The performance of the RoamML model in context-aware is presented in Fig. 7.7a, as previously explained two line to depict the accuracy and the loss This ordering illustrates



(a) Accuracy Metrics



(b) Forgetting Metrics

Figure 7.6: Context-Aware scenario RoamML performance. (a) **Accuracy Metric:** Tracks model accuracy and loss across the sequence of nodes over time. The horizontal yellow line indicates the centralized CNN baseline (83% accuracy). (b) **Forgetting Metric:** Illustrates the model's forgetting metrics across training, with individual forgetting values (blue) and the average forgetting trend (red). We can observe a slight drop in performance at node 13, which is attributed to its highly unbalanced data distribution. This distributional shift increases the difficulty of adaptation and leads to higher forgetting.

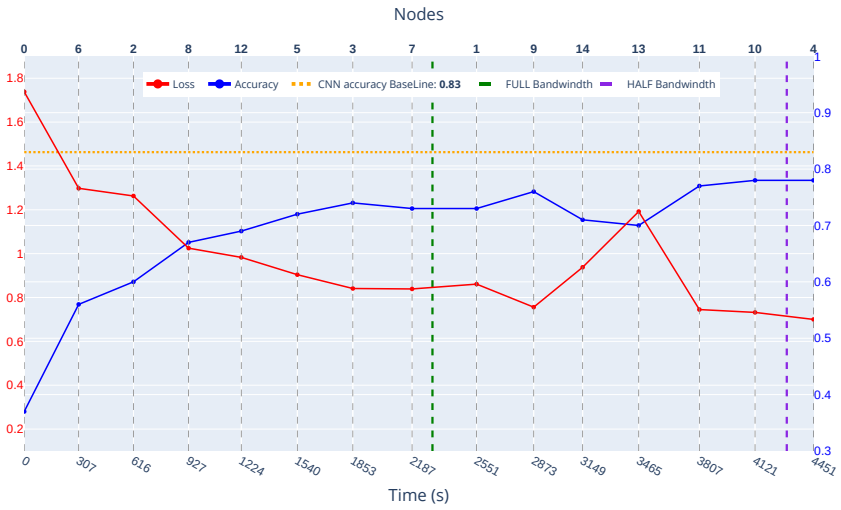
the strategic decisions made by the Agent in prioritizing node visits via MCDM based on the combo of *Entropy-TOPSIS*. The **test loss reached 0.7**. On the other hand, we can note a steady increase in the test reaching an **accuracy of 78%**. As in Fig. 7.7b, we can observe the Forgetting Metrics and the **Average Forgetting** at the end of the trip in this scenario was approximately **-0.0186**.

7.5.6 Final Considerations and Observations

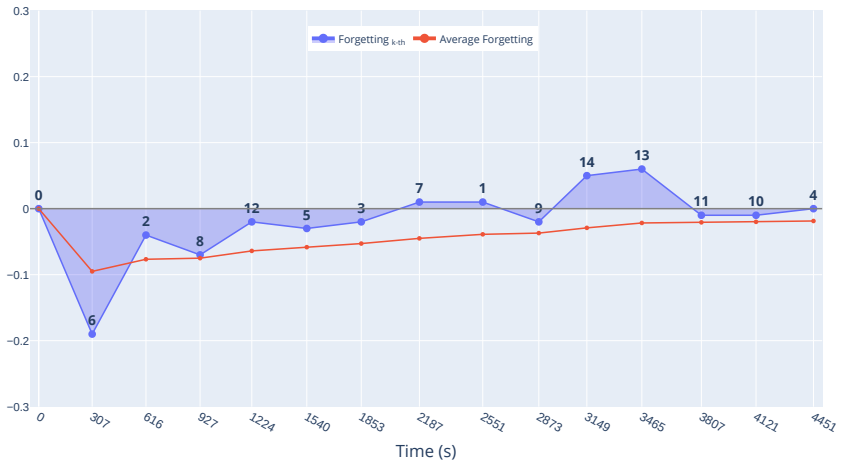
The results summarized in Table 7.5 confirm the validity of the RoamML model, which can provide such a remarkable performance in terms of accuracy and network bandwidth consumption even when datasets are highly diverse and imbalanced. Although the original RoamML DG model used much more bandwidth if we compare it to the newly developed DG Model. All the RoamML based scenarios have managed to **save up to a significant 75% in bandwidth** compared to the centralized baseline scenario. In terms of ML performance, the accuracy on the test set of RoamML on both context-aware and independent, which are just approximately 5 points of accuracy away from the centralized baseline trained model, which saturates at 83% of accuracy. The Average Forgetting can also be considered, where both scenarios have shown negative forgetting, thus showcasing that the model is acquiring knowledge. Nonetheless, a context-aware scenario (-0.0293) has shown better results in terms of Average Forgetting than context-independent (-0.0186), where a greater weight was put on the right criteria (e.g. Data distribution) the model managed to forget less and accumulate more knowledge in its journey.

Code Availability: Documented code that can be used to reproduce or build upon the experiments and framework is available online: https://github.com/DSG-UniFE/RoamML.

We further assessed the final trained model in terms of deployability and resource



(a) Accuracy Metrics



(b) Forgetting Metrics

Figure 7.7: Context-Indipendent scenario RoamML performance. (a) **Accuracy Metric:** Tracks model accuracy and loss across the sequence of nodes over time. The horizontal yellow line indicates the centralized CNN baseline (83% accuracy). (b) **Forgetting Metric:** Illustrates model's forgetting metrics across training, with individual forgetting values (blue) and the average forgetting trend (red). We can observe a noticeable drop in performance at nodes 13 and 14, which is attributed to their highly skewed and unbalanced data distributions. This distributional shift increases the difficulty of adaptation and leads to higher forgetting, particularly when these nodes are visited consecutively.

Table 7.5: Comparison of Different RoamML Models

Metrics	*RoamML*			*Centralized Baseline*	*Federated Learning*
	Single Criteria	*Aware*	*Independent*		
Avg. Forget	-0.0239	**-0.0293**	-0.0186	N/A	N/A
Accuracy	76%	**78%**	**78%**	83%	80%
Loss	0.758	0.716	**0.70**	0.51	0.72
Time (s)	4580	4482	**4451**	2008	N/A
Bandwidth (MB)	1244	**789**	931	4141	N/A

efficiency. Thanks to the consistent use of a lightweight architecture across all experiments, the model size remained compact at just 6.5 MB, well-suited for edge deployment. Naturally, this depends on both the dataset and the specific model configuration; in our case, the footprint also reflects the replay buffer size (250 images $\approx 225 \times 3KB = 0.73MB$) along with additional files and scripts on the order of a few kilobytes. The total size of the deployed agent is therefore approximately $7.5MB$ on average. To evaluate real-world feasibility, we deployed the final model on a Raspberry Pi 4 (with 2 GB RAM and 16 GB of storage), a representative edge computing device, and measured inference latency. The average inference time was 0.21 seconds per image across 1,000 test cycles, confirming that the model is capable of supporting near real-time decision-making in the field.

These results are particularly noteworthy given that the ML model was trained in a distributed continual learning fashion on partial data subsets, without requiring full data centralization. In the baseline scenario, transferring all data to a centralized hub proved to be highly taxing on the network, consuming a very significant bandwidth. Despite this, the RoamML model achieved performance levels close to that of the centralized approach, demonstrating that it can deliver comparable ML outcomes while drastically reducing network load. This comparison underscores the effectiveness of the RoamML

framework, which not only matches centralized training in accuracy but also excels in communication efficiency, making it a practical and scalable solution for bandwidth-constrained environments.

Nonetheless, while the presented results demonstrate the effectiveness of the proposed approach, the current evaluation focuses primarily on representative scenarios and average performance metrics. Due to the complexity of distributed continual learning under dynamic connectivity conditions, repeated trials under identical configurations are not always deterministic, as node availability, network conditions, and data arrival patterns may vary. As a result, the analysis emphasizes comparative performance trends and communication efficiency rather than formal statistical confidence intervals. Future work will extend this evaluation by conducting multiple experimental repetitions across diverse network configurations and stochastic conditions, enabling the computation of variance, confidence intervals, and statistical significance measures. This will provide a more comprehensive assessment of the robustness, stability, and generalizability of the RoamML framework under varying operational environments.

7.6 Chapter Summary

This chapter examined Type 3 Disaster-Impacted Environments (DDIL environments), where communication infrastructures are degraded, intermittent, or absent. Such conditions pose unique challenges for machine learning, requiring adaptive approaches that can withstand connectivity collapse, extreme bandwidth scarcity, fragile edge hardware, and constrained power supplies while sustaining distributed data-driven operations.

These considerations motivated the design of RoamML, a novel framework for distributed continual learning. RoamML follows the principle that moving a model is more efficient and resilient than transferring large datasets or exchanging frequent parameter updates. It integrates continual learning algorithms to enable incremental updates while mitigating catastrophic forgetting. Its agent-based architecture assigns models to roaming agents that traverse network nodes, performing on-site training and adapting dynamically to evolving conditions.

At the core of RoamML lies the concept of Data Gravity, which directs the roaming agent's next-hop decisions. Originally defined as a simple dataset-size heuristic, Data Gravity was extended in this chapter into a multi-dimensional model that incorporates dataset characteristics, resource constraints, and model performance indicators. By embedding Multi-Criteria Decision-Making (MCDM) processes, RoamML balances objective metrics (e.g., bandwidth, connectivity, dataset size) with subjective expert-driven criteria (e.g., urgency, mission relevance). This enables the framework to adapt flexibly to evolving disaster response priorities. We validated RoamML through extensive experimentation in an emulated environment. Comparisons with centralized and federated learning approaches demonstrated that RoamML achieves accuracy comparable to centralized ML in ideal conditions, while substantially outperforming both baselines in bandwidth-constrained scenarios, reducing bandwidth consumption by up to 75%.

In conclusion, this chapter showed that RoamML provides a resilient, adaptive frame-

work for machine learning in disaster-impacted settings, showing that novel and adaptive ML frameworks are not only feasible but essential under extreme conditions.

Chapter 8

Summary of Research Outputs and Contributions

During this doctoral research, I achieved several significant academic contributions that demonstrate both scientific impact and practical relevance. My work led to the publication of six peer-reviewed journal articles in prestigious venues, including Elsevier's *Computers in Industry* (Q1) [63], Elsevier's *Journal of Network and Computer Applications* (Q1) [84], Springer's *Data Science and Engineering* (Q1) [22], Elsevier's *Internet of Things* (Q1) [103], IEEE's *Transactions on Industrial Informatics* (Q1) [79], and the *Infocommunications Journal* (Q2) [20]. In addition to journal publications, I authored thirteen international conference papers disseminated across leading venues in distributed systems, artificial intelligence, and industrial informatics.

Two of these contributions received formal recognition through Best Paper Awards. The first award was granted at the 2023 International Conference on Military Communications and Information Systems (ICMCIS), in collaboration with the NATO IST 194 Research Task Group, for the paper titled "A Data Mesh Approach for Enabling Data-

Centric Applications at the Tactical Edge" [5]. The second was awarded at the 2024 Italian Conference on Big Data and Data Science (ITADATA) for the paper "Multivariate Time Series Anomaly Detection in Industry 5.0" [21]. These recognitions underscore the novelty and scientific rigor of the proposed methods, as well as their relevance in both military and industrial contexts.

Beyond academic publishing, the Ph.D. involved extensive collaboration with four international manufacturing companies through the funded projects "Data-driven IT Services for Sustainable and Efficient Manufacturing (DISSEM)" and "Next Generation Analytics for Manufacturing (NG4M)". These collaborations enabled the development, deployment, and evaluation of Machine Learning and MLOps solutions directly within real industrial environments. As a result of this applied work, the research produced one international patent with the Carpigiani Group [104] and a second patent application currently under review. These outcomes demonstrate the technological maturity and industrial transferability of the research presented in this manuscript.

The doctoral journey also offered the opportunity to supervise seven brilliant Master's students, guiding their theses in areas such as anomaly detection, MLOps architectures, and industrial AI systems. I additionally mentored numerous Bachelor's students during their internships at the IN4 Hub (formerly MechLav), one of the industrial research laboratories of the Tecnopolo di Ferrara. This engagement in mentoring and supervision contributed to the training of the next generation of engineers and researchers in data-driven manufacturing.

The research benefited greatly from continuous collaboration with Dr. Niranjan Suri, leader of the NOMADS group. It included a 6+ months research period as an exchange scholar at the Institute for Human & Machine Cognition (IHMC) in Pensacola, Florida, USA. Together, these experiences shaped a multidisciplinary and internationally oriented research path that resulted in tangible scientific, technological, and industrial contributions.

Chapter 9

Conclusions & Future Work

This work presents a systematic investigation into the use of Machine Learning in High-Stakes Environments, in particular, it presents a typology framework that categorises High-Stakes Environments according to key connectivity characteristics – reliability, bandwidth, and latency – and links these directly to their implications for ML workflows, particularly in training, deployment, and long-term system resilience. Building on this framework, the dissertation examined three representative real-world case studies, each aligned with one environment type and illustrating how the proposed classification can guide engineering decisions in practice.

The first case study, situated in a smart manufacturing environment, demonstrated how robust, stable infrastructure enables large-scale ML integration with minimal compromise. The availability of high-bandwidth, low-latency connectivity and significant computational resources facilitated advanced data collection pipelines, end-to-end ML workflow automation. The second case study, centered on in-field industrial environments, highlighted the constraints imposed by constrained connectivity and limited edge hardware. These limitations required hybrid cloud-edge strategies for both training and

deployment of the presented HoT-AI system for the ice cream machines, making them more adaptable to the changes in the used recipes. Through the development of the HoT-AI system, the study demonstrated how ML workflows must adapt dynamically to infrastructural variability, ensuring that inference remains timely and robust even when cloud access is limited. The third case study, focused on disaster-impacted environments, illustrated the necessity of adaptive ML solutions under severely degraded or fully disconnected infrastructures. These contexts require systems capable of autonomous operation and learning despite uncertainty, instability, and delayed or absent communication. To address these challenges, the dissertation introduced RoamML, a novel distributed continual learning framework based on mobile intelligent agents guided by data gravity principles. Through extensive experimentation, RoamML was shown to outperform centralized and federated approaches when infrastructure is unreliable, reaffirming the importance of designing ML solutions that are intrinsically resilient to connectivity degradation. This final case study provided empirical evidence for the typology's most demanding category, demonstrating how its principles can inform the creation of fundamentally new workflow paradigms.

Across all three case studies, the typology framework proved to be a valuable interpretive and design tool, enabling engineers to assess constraints systematically, anticipate failure modes, and select ML strategies that align with operational realities. By articulating how connectivity shapes the entire machine learning life cycle, the work contributes both conceptual clarity and practical guidance to a domain where decisions often span technical, organizational, and environmental dimensions. This work summarises the research conducted during my Ph.D. at the Department of Engineering of the University of Ferrara. Throughout this journey, the overarching goal has been to bridge the gap between Machine Learning theory and the operational constraints of real-world High-Stakes Environments. By proposing a unifying framework, validating it through heterogeneous practical applications, and developing novel methodologies tailored to

extreme conditions, the dissertation contributes a structured pathway for designing resilient, efficient, and context-aware ML solutions.

Future work may extend this research in several directions. One particularly interesting avenue concerns energy-aware Machine Learning, especially in light of the rapid emergence and widespread deployment of large and foundation models. These models introduce substantial computational and memory demands, which translate directly into significant energy consumption – both during training and inference. While this dissertation focused primarily on connectivity-related constraints, energy availability is increasingly becoming a critical limiting factor in High-Stakes Environments, influencing not only feasibility but also environmental impact, and long-term sustainability. Advancing this line of research will require a deeper understanding of how energy constraints intersect with ML workflows across different environment types. For instance, in partially connected or fully disconnected settings, energy scarcity may exacerbate the challenges already posed by limited bandwidth and high latency, forcing ML systems to rely on ultra-lightweight models or adaptive scheduling mechanisms that match computational effort to available power. At the other end of the spectrum, even resource-rich infrastructures face mounting pressure to reduce their carbon footprint, suggesting an opportunity to explore cost–benefit frameworks that jointly consider predictive performance, operational risk, and energy efficiency. Future investigations could therefore evaluate an energy-aware model, integrate power-profiling into MLOps pipelines, or extend the typology framework to incorporate energy as a fourth structural dimension. Incorporating energy constraints into system design would substantially strengthen the ability of ML systems to operate reliably in high-stakes contexts, where sustainability, autonomy, and safety are deeply intertwined.

Another important aspect concerns security, which is essential for the reliable deployment of Machine Learning systems in High-Stakes Environments. In these contexts, ML models directly influence operational decisions, making them potential targets for

adversarial attacks, data poisoning, or model tampering. Such threats can compromise system integrity and lead to unsafe or incorrect behavior, particularly in distributed or partially connected environments where centralized oversight is limited. Future work should integrate security within the proposed typology framework. This includes developing secure data pipelines, robust and tamper-resistant models, and trust-aware distributed learning mechanisms. In frameworks such as RoamML, incorporating trust evaluation and secure model exchange between nodes could further enhance resilience. Addressing these challenges will be essential to ensuring that ML systems remain trustworthy, robust, and safe in high-stakes operational settings.

Appendices

Appendix A: Deep Learning Common Functions

This appendix summarizes commonly used functions in deep learning models, including their mathematical formulations and typical use cases.

1. Sigmoid Function

The sigmoid function outputs values in the range $[0, 1]$ and is often used in binary classification problems, particularly in the output layer.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{9.1}$$

2. Hyperbolic Tangent (Tanh)

The hyperbolic tangent function outputs values in the range $[-1, 1]$. It is frequently used in hidden layers due to its zero-centered nature, which can lead to faster convergence.

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{9.2}$$

3. Rectified Linear Unit (ReLU)

The ReLU function is widely used for its simplicity and computational efficiency. It helps mitigate the vanishing gradient problem during training.

$$\text{ReLU}(x) = \max(0, x) \tag{9.3}$$

4. Softmax

The softmax function converts a vector of real-valued scores z into a probability distribution over M classes. It is commonly used in the output layer for multi-class classification tasks.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{M} e^{z_j}} \tag{9.4}$$

Appendix B: Loss Functions

Loss functions quantify the discrepancy between predicted values $\hat{y}_i$ and actual target values y_i. They serve as the objective to be minimized during training.

Regression Loss Functions

- **Mean Squared Error (MSE)**: Measures the average of the squares of the errors.

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{9.5}$$

- **Mean Absolute Error (MAE)**: Measures the average of the absolute differences.

$$\mathcal{L}_{\text{MAE}} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \tag{9.6}$$

- **Root Mean Squared Error (RMSE)**: The square root of MSE, more sensitive to large errors.

$$\mathcal{L}_{\text{RMSE}} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{9.7}$$

Classification Loss Functions

- **Cross Entropy Loss (CEL)**: Measures the performance of a classification model whose output is a probability value between 0 and 1.

$$\mathcal{L}_{\text{CE}} = -\sum_{c=1}^{M} y_c \log(p_c) \tag{9.8}$$

  Where:

  - M is the number of classes,
  - y_c is a binary indicator (1 if class c is the correct class, 0 otherwise),
  - p_c is the predicted probability of class c.

Autoencoder Reconstruction Error

In autoencoders, the objective is to learn an efficient representation (encoding) of the input data such that the reconstructed output closely approximates the original input. The difference between the input $\mathbf{x}$ and its reconstruction $\hat{\mathbf{x}}$ is measured by the reconstruction error.

For a dataset of N samples, the reconstruction error is typically calculated using the Mean Squared Error (MSE):

$$\mathcal{L}_{\text{recon}} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \tag{9.9}$$

Where:

- $\mathbf{x}_i$ is the original input vector for the i-th sample,
- $\hat{\mathbf{x}}_i$ is the reconstructed output from the autoencoder,
- $\| \cdot \|^2$ denotes the squared Euclidean norm.

This loss encourages the network to minimize the difference between inputs and outputs, effectively learning a compact and informative latent representation of the data.

Appendix C: Distance Metrics

This appendix presents commonly used distance metrics in machine learning, particularly for evaluating similarity between vectors or time-series data.

1. L2 Distance (Euclidean Distance)

L2 distance is one of the most widely used metrics for measuring the distance between two vectors in Euclidean space. Given two vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$, the L2 distance is defined as:

$$D_{\text{L2}}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_2 = \sqrt{\sum_{i=1}^{d} (x_i - y_i)^2} \tag{9.10}$$

Where:

- $\mathbf{x} = (x_1, x_2, \ldots, x_d)$

- $\mathbf{y} = (y_1, y_2, \ldots, y_d)$
- d is the dimensionality of the vectors

It is commonly used in clustering, nearest-neighbor search, and anomaly detection.

2. Dynamic Time Warping (DTW)

Dynamic Time Warping (DTW) is a powerful algorithm used to measure the similarity between two time series that may vary in speed or length. It aligns sequences by warping the time axis to find an optimal match between them.

Given two sequences:

$$\mathbf{x} = (x_1, x_2, \ldots, x_n), \quad \mathbf{y} = (y_1, y_2, \ldots, y_m) \tag{9.11}$$

The DTW distance is computed by constructing an $n \times m$ cost matrix where each element is:

$$D(i,j) = (x_i - y_j)^2 + \min \begin{cases} D(i-1,j), \\ D(i,j-1), \\ D(i-1,j-1) \end{cases} \tag{9.12}$$

With the base condition:

$$D(0,0) = 0, \quad D(i,0) = D(0,j) = \infty \text{ for } i,j > 0 \tag{9.13}$$

The final DTW distance is:

$$\text{DTW}(\mathbf{x}, \mathbf{y}) = \sqrt{D(n,m)} \tag{9.14}$$

DTW is especially useful in speech recognition, gesture analysis, and sensor data

alignment.

Appendix D: Classification Performance Metrics

In classification problems, especially with imbalanced data, relying solely on accuracy can be misleading. Therefore, a variety of performance metrics are used to evaluate different aspects of model behavior. Let TP, FP, FN, and TN denote true positives, false positives, false negatives, and true negatives, respectively.

Accuracy

Accuracy measures the overall correctness of the classifier by calculating the proportion of correctly predicted instances (both positive and negative) over the total number of instances:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{9.15}$$

While accuracy is intuitive and useful in balanced datasets, it may obscure poor performance on the minority class in imbalanced scenarios.

Precision

Precision quantifies how many of the instances predicted as positive are actually positive:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{9.16}$$

Recall (Sensitivity)

Recall measures how many actual positive instances are correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{9.17}$$

F_β Score

The F_β-score is the weighted harmonic mean of Precision and Recall. The parameter $\beta > 0$ determines the weight of Recall relative to Precision:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{(\beta^2 \cdot \text{Precision}) + \text{Recall}} \tag{9.18}$$

Special cases include F_1 (balanced), F_2 (Recall-focused), and $F_{0.5}$ (Precision-focused).

Specificity

Specificity, or true negative rate, measures the proportion of correctly predicted negatives:

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{9.19}$$

Geometric Mean (G-Mean)

G-Mean assesses the balance between sensitivity and specificity, especially in imbalanced datasets:

$$\text{G-Mean} = \sqrt{\text{Recall} \cdot \text{Specificity}} = \sqrt{\frac{TP}{TP + FN} \cdot \frac{TN}{TN + FP}} \tag{9.20}$$

These metrics together provide a comprehensive evaluation framework, particularly suited for high-stakes or imbalanced classification tasks.

Complexity Analysis of Node-Ordering Calculation

Entropy Weighting Method

Suppose there are m criteria and n alternatives. Let x_{ij} be the measured value of the i-th criterion in the j-th alternative, where $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, n$. The EWM procedure is outlined in the following steps [57]:

Step 1: Standardization of the Measured Values To obtain a standardized value p_{ij}, each x_{ij} is normalized by the sum of all x_{ij} over the alternatives:

$$p_{ij} = \frac{x_{ij}}{\sum_{j=1}^{n} x_{ij}}. \tag{1}$$

This step requires iterating over all $m \times n$ entries, leading to a time complexity of:

$$O(m \times n)$$

Step 2: Calculation of Entropy Next, the entropy value E_i for each criterion i is computed as:

$$E_i = -\frac{1}{\ln(n)} \sum_{j=1}^{n} p_{ij} \ln\big(p_{ij}\big), \tag{2}$$

with the convention that $p_{ij} \ln(p_{ij}) = 0$ when $p_{ij} = 0$. Computing the summation for each criterion involves n operations; hence, for all m criteria, this step has a time complexity of:

$$O(m \times n)$$

Step 3: Determination of Weights After calculating the entropy values, the weight w_i for each criterion is determined by first computing the information utility $1 - E_i$ and

then normalizing:

$$w_i = \frac{1 - E_i}{\sum_{i=1}^{m} (1 - E_i)}. \tag{3}$$

The complexity of this step is dominated by the sum over m terms and basic arithmetic operations, hence it is:

$$O(m)$$

Overall Time Complexity Summing up the complexities of all steps:

$$O(m \times n) \; + \; O(m \times n) \; + \; O(m) \; = \; O(m \times n)$$

Thus, the dominant factor in the entropy-based weighting algorithm is the $m \times n$ term, and the overall time complexity is:

$$O(m \times n)$$

Best Worst Method (BWM) for Weighting Criteria

Assume there are m criteria denoted by $C = \{c_1, c_2, \ldots, c_m\}$. The Best Worst Method (BWM) [59] involves the following steps:

Step 1: Identification of Best and Worst Criteria A decision maker selects:

- The **best** (most important) criterion, denoted c_B.
- The **worst** (least important) criterion, denoted c_W.

Identifying the best and worst criteria that are provided by the decision maker is:

$$O(1)$$

Step 2: Elicitation of Preferences The decision maker provides two sets of pairwise comparison values based on a predetermined scale (typically 1 to 9):

1. **Best-to-Others Comparison:** Evaluate the preference of the best criterion over every other criterion:

$$a_{Bj}, \quad \text{for } j = 1, 2, \ldots, m,$$

with $a_{BB} = 1$

2. **Others-to-Worst Comparison:** Evaluate the preference of each criterion over the worst criterion:

$$a_{jW}, \quad \text{for } j = 1, 2, \ldots, m,$$

with $a_{WW} = 1$

In forming the comparison vectors "Best-to-Others" and "Others-to-Worst", we have $m - 1$ comparison for each. Therefore, the algorithm requires a total of $2(m - 1)$ comparisons, minus one since the best and worst criteria are compared only once, resulting in $2m - 3$ comparisons [105]. The overall complexity of this step is:

$$O(m)$$

Step 3: Formulation of the Optimization Model The optimal weights $w_1, w_2, \ldots, w_m$ are obtained by solving the following minimax optimization problem:

$$\min_{\xi} \quad \xi \tag{9.21}$$

$$\text{subject to} \quad \left| \frac{w_B}{w_j} - a_{Bj} \right| \leq \xi, \quad \forall\, j = 1, 2, \ldots, m, \tag{9.22}$$

$$\left| \frac{w_j}{w_W} - a_{jW} \right| \leq \xi, \quad \forall\, j = 1, 2, \ldots, m, \tag{9.23}$$

$$\sum_{j=1}^{m} w_j = 1, \quad w_j \geq 0, \quad \xi \geq 0. \tag{9.24}$$

In this model, the variable ξ represents the maximum deviation between the weight ratios and the provided preference comparisons. The goal is to minimize ξ so that the derived weights best reflect the decision maker's judgments.

Step 4: Solving the Model The optimization problem above is typically linearized and solved using a linear programming (LP) solver. The resulting optimal solution provides the weight vector:

$$\mathbf{w} = (w_1, w_2, \ldots, w_m)$$

The overall time complexity of the BWM weighting process comprises the following components: The LP model includes m variables (the weights) and approximately $2m$ inequality constraints. Using an interior-point method or similar algorithm, the worst-case time complexity to solve the LP is typically polynomial in m. In practice, this is often approximated as $O(m^3)$ as illustrated in [106]. Therefore, the overall time complexity for the Best Worst Method is dominated by the LP solution phase, yielding:

$$O(m^3)$$

TOPSIS

The process of computing the optimal order in which to visit candidate nodes is based on a multi-criteria decision-making (MCDM) algorithm. In our approach, this involves normalising evaluation criteria, computing individual weights (using techniques such as entropy or Best Worst Method), and then using TOPSIS to aggregate and order nodes from best to worst. The time complexity of this procedure is $O(NM)$, where N represents the number of alternative nodes and M represents the number of evaluation criteria. In our specific scenario, the number of alternatives (e.g., nodes) is limited—typically on the order of 15—and the evaluation is based on a relatively small set of criteria (e.g., 8 criteria). This results in a very modest computational burden. The TOPSIS algorithm

that we opted for can be broken down into several stages, each contributing to the overall time complexity [107].

Stage 1: Normalization and Weighting In this stage, we normalize the decision matrix and then apply the corresponding weights. For a decision matrix of size $n \times m$, this process requires iterating through all entries, resulting in a time complexity of:

$$O(n \times m)$$

Stage 2: Identification of Ideal Solutions Next, the positive ideal solution (PIS) and negative ideal solution (NIS) are determined for each criterion. Assuming there are m criteria, this step involves scanning through the alternatives for each criterion, which has a complexity on the order of:

$$O(m)$$

(In some descriptions, it might be stated as $O(n)$ depending on the data structure, but typically the criteria dimension m is the focus for this stage.)

Stage 3: Distance Calculation For each alternative, the algorithm computes the distance to both the PIS and the NIS. This step generally runs in $O(n)$, since each alternative is processed independently.

Stage 4: Final Ranking Finally, the algorithm computes the relative closeness or ranking of each alternative. If the ranking involves sorting the n alternatives, the cost of this step is:

$$O(n \log n)$$

Total Time Complexity Summing up all the operations, the total time complexity is:

$$O(n \times m) + O(m) + O(n) + O(n \log n)$$

Since $O(n \times m)$ is the dominant term for large n and m, the overall time complexity can be expressed as:

$$O(n \times m)$$

Considerations of Complexity Analysis

In a **context-aware scenario** using the Best-Worst Method (BWM) with TOPSIS, the complexity analysis of the node-ordering calculation is as follows:

$$O(m^3) \ + \ O(m \times n) \ = \ O(m^3),$$

For a **context-independent scenario** where the Entropy method is used with TOPSIS, the total complexity is:

$$O(m \times n) \ + \ O(m \times n) \ = \ O(m \times n).$$

Overall, these analyses highlight the trade-offs between the two approaches: the context-aware scenario incurs a higher computational cost due to the additional optimization step in the BWM, while the context-independent scenario remains more scalable with a linear dependence on $m \times n$. This makes the Entropy method more efficient for larger scenarios, particularly when there are many more alternatives to evaluate and efficiency is critical. Nonetheless, since the MCDM computation is not time-critical—given that it operates on a limited number of nodes and criteria—it does not introduce significant latency into the overall operational workflow. The time required for ordering the nodes based on the computed gravity scores is minimal compared to the duration of

the machine learning model training (performed in parallel to compute the node order) and the data transfer times between nodes. This allows the system to reserve the majority of computational resources for intensive training and communication tasks, ensuring that the path determination process remains a lightweight component of the overall system performance.

Additional Note: In our case, the number of nodes and alternatives is very small. For example, with 15 alternatives and 8 criteria, the computational time is negligible compared to the training times of the ML models and the transfer times of the agent.

References

[1] A. Frankó, G. Hollósi, D. Ficzere, and P. Varga, "Applied machine learning for iiot and smart production—methods to improve production quality, safety and sustainability," *Sensors*, vol. 22, no. 23, 2022, ISSN: 1424-8220.

[2] A. E. Frankó and P. Varga, "A survey on machine learning based smart maintenance and quality control solutions," *Infocommunications Journal*, vol. 13, no. 4, pp. 28–35, 2021.

[3] S. Zeb, A. Mahmood, S. A. Khowaja, *et al.*, "Towards defining industry 5.0 vision with intelligent and softwarized wireless network architectures and services: A survey," *Journal of Network and Computer Applications*, vol. 223, p. 103 796, 2024, ISSN: 1084-8045. DOI: 10.1016/j.jnca.2023.103796.

[4] S. Dahdal and M. Tortonesi, "Enabling big data and machine learning applications in high-stakes environments," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, IEEE, May 2024, pp. 1–4. DOI: 10.1109/noms59830.2024.10574906.

[5] S. Dahdal, F. Poltronieri, M. Tortonesi, C. Stefanelli, and N. Suri, "A data mesh approach for enabling data-centric applications at the tactical edge," in *2023 In-*

ternational Conference on Military Communications and Information Systems (ICMCIS), 2023, pp. 1–9. DOI: 10.1109/ICMCIS59922.2023.10253568.

[6] L. Colombi, A. Gilli, S. Dahdal, *et al.*, "A machine learning operations platform for streamlined model serving in industry 5.0," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–6. DOI: 10.1109/NOMS59830.2024.10575103.

[7] B. Sahoh and A. Choksuriwong, "The role of explainable artificial intelligence in high-stakes decision-making systems: A systematic review," *Journal of Ambient Intelligence and Humanized Computing*, vol. 14, no. 6, pp. 7827–7843, Apr. 2023. DOI: 10.1007/s12652-023-04594-w.

[8] A. Frankó, G. Hollósi, D. Ficzere, and P. Varga, "Applied machine learning for iiot and smart production—methods to improve production quality, safety and sustainability," *Sensors*, vol. 22, no. 23, p. 9148, 2022.

[9] L. Campioni, F. Poltronieri, C. Stefanelli, N. Suri, M. Tortonesi, and K. Wrona, "Enabling civil–military collaboration for disaster relief operations in smart city environments," *Future Generation Computer Systems*, vol. 139, pp. 181–195, 2023.

[10] R. R. Arinta and E. Andi W.R., "Natural disaster application on big data and machine learning: A review," in *2019 4th International Conference on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, 2019, pp. 249–254. DOI: 10.1109/ICITISEE48480.2019.9003984.

[11] M. Yu, C. Yang, and Y. Li, "Big data in natural disaster management: A review," *Geosciences*, vol. 8, no. 5, 2018, ISSN: 2076-3263. DOI: 10.3390/geosciences8050165.

[12] C. Sicari, A. Catalfamo, L. Carnevale, *et al.*, "Tema: Event driven serverless workflows platform for natural disaster management," in *2023 IEEE Sympo-*

sium on Computers and Communications (ISCC), 2023, pp. 1–6. DOI: 10.1109/ISCC58397.2023.10217920.

[13] V. Chamola, V. Hassija, S. Gupta, A. Goyal, M. Guizani, and B. Sikdar, "Disaster and pandemic management using machine learning: A survey," *IEEE Internet of Things Journal*, vol. 8, no. 21, pp. 16 047–16 071, 2021. DOI: 10.1109/JIOT.2020.3044966.

[14] M. S. Hackathon, *Leveraging ai for natural disaster management : Takeaways from the moroccan earthquake*, 2023. arXiv: 2311.08999 [cs.AI].

[15] R. Banomyong, P. Varadejsatitwong, and R. Oloruntoba, "A systematic review of humanitarian operations, humanitarian logistics and humanitarian supply chain performance literature 2005 to 2016," *Annals of Operations Research*, vol. 283, no. 1–2, pp. 71–86, 2017. DOI: 10.1007/s10479-017-2549-5.

[16] H. Shiri, J. Park, and M. Bennis, "Communication-efficient massive uav online path control: Federated learning meets mean-field game theory," *IEEE Transactions on Communications*, vol. 68, no. 11, pp. 6840–6857, 2020. DOI: 10.1109/TCOMM.2020.3017281.

[17] H. Kunreuther, R. Meyer, R. Zeckhauser, *et al.*, "High stakes decision making: Normative, descriptive and prescriptive considerations," *Marketing Letters*, vol. 13, no. 3, pp. 259–268, Aug. 2002, ISSN: 1573-059X. DOI: 10.1023/a:1020287225409.

[18] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, 2004. DOI: 10.1109/TDSC.2004.2.

[19] A. Géron, *Hands-on machine learning with Scikit-Learn and TensorFlow concepts, tools, and techniques to build intelligent systems*, 2nd ed. O'Reilly Media, Inc., 2019, ISBN: 9781492032649.

[20] S. Dahdal, L. Colombi, M. Brina, *et al.*, "An mlops framework for gan-based fault detection in bonfiglioli's evo plant," *Infocommunications journal*, vol. 16, no. 2, pp. 2–10, 2024, ISSN: 2061-2079. DOI: 10.36244/icj.2024.2.1.

[21] L. Colombi, M. Vespa, N. Belletti, *et al.*, *Multivariate time series anomaly detection in industry 5.0*, 2025. arXiv: 2503.15946 [cs.LG].

[22] L. Colombi, M. Vespa, N. Belletti, *et al.*, "Embedding models for multivariate time series anomaly detection in industry 5.0," *Data Science and Engineering*, Jun. 2025, ISSN: 2364-1541. DOI: 10.1007/s41019-025-00295-w.

[23] R. Galliera, A. Morelli, R. Fronteddu, and N. Suri, "Marlin: Soft actor-critic based reinforcement learning for congestion control in real networks," in *NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium*, 2023, pp. 1–10. DOI: 10.1109/NOMS56928.2023.10154210.

[24] L. Colombi, M. Brina, M. Vespa, *et al.*, "Optimizing industry 5.0 machine learning-based applications via synthetic data generation," in *2024 IEEE 29th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2024, pp. 1–6. DOI: 10.1109/CAMAD62243.2024.10942898.

[25] L. Ruthotto and E. Haber, *An introduction to deep generative modeling*, May 2021. DOI: 10.1002/gamm.202100008.

[26] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002, ISSN: 1076-9757. DOI: 10.1613/jair.953.

[27] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, 2008, pp. 1322–1328. DOI: 10.1109/IJCNN.2008.4633969.

[28] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15, Lille, France: JMLR.org, 2015, pp. 2256–2265.

[29] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 6840–6851.

[30] A. Kotelnikov, D. Baranchuk, I. Rubachev, and A. Babenko, "Tabddpm: Modelling tabular data with diffusion models," in *International Conference on Machine Learning*, PMLR, 2023, pp. 17 564–17 579.

[31] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, "Modeling tabular data using conditional gan," *Advances in neural information processing systems*, vol. 32, 2019.

[32] Y. Li, Q. Wang, J. Zhang, L. Hu, and W. Ouyang, "The theoretical research of Generative Adversarial Networks: an overview," *Neurocomputing*, vol. 435, pp. 26–41, 2021, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2020.12.114.

[33] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learn-*

ing, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, Aug. 2017, pp. 214–223.

[34] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *Journal of Network and Computer Applications*, vol. 116, pp. 1–8, 2018, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2018.05.003.

[35] L. Colombi, E. D. Caro, S. Dahdal, *et al.*, "Fededge-learn: A semi-supervised federated learning framework for industry 5.0," in *2025 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2025, pp. 1–6. DOI: 10.1109/ISCC65549.2025.11326181.

[36] J. Li, X. Liu, and T. Mahmoodi, "Federated learning in heterogeneous wireless networks with adaptive mixing aggregation and computation reduction," *IEEE Open Journal of the Communications Society*, vol. 5, pp. 2164–2182, 2024. DOI: 10.1109/OJCOMS.2024.3381545.

[37] D. Wang, S. Guan, and R. Sun, "A novel staged training strategy leveraging knowledge distillation and model fusion for heterogeneous federated learning," *Journal of Network and Computer Applications*, vol. 236, p. 104 104, 2025, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2025.104104.

[38] F. Dong, H. Leung, and S. Drew, "Optimizing federated learning with weighted aggregation in aerial and space networks," *Journal of Network and Computer Applications*, vol. 235, p. 104 086, 2025, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2024.104086.

[39] W. Li, H. Hacid, E. Almazrouei, and M. Debbah, "A comprehensive review and a taxonomy of edge machine learning: Requirements, paradigms, and techniques," *AI*, vol. 4, no. 3, pp. 729–786, Sep. 2023, ISSN: 2673-2688. DOI: 10.3390/ai4030039.

[40] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE Transactions on Information Theory*, vol. 52, no. 6, pp. 2508–2530, 2006. DOI: 10.1109/TIT.2006.874516.

[41] A. Douillard, Q. Feng, A. A. Rusu, *et al.*, *Diloco: Distributed low-communication training of language models*, 2023. arXiv: 2311.08105 [cs.LG].

[42] L. Wang, X. Zhang, H. Su, and J. Zhu, *A comprehensive survey of continual learning: Theory, method and application*, 2023. arXiv: 2302.00487 [cs.LG].

[43] S.-s. Zhang, J.-w. Liu, and X. Zuo, "Adaptive online incremental learning for evolving data streams," *Applied Soft Computing*, vol. 105, p. 107 255, 2021, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2021.107255.

[44] T. Bouvier, B. Nicolae, A. Costan, T. Bicer, I. Foster, and G. Antoniu, "Efficient distributed continual learning for steering experiments in real-time," *Future Generation Computer Systems*, vol. 162, p. 107 438, 2025, ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2024.07.016.

[45] F. Ye and A. G. Bors, "Continual compression model for online continual learning," *Applied Soft Computing*, vol. 167, p. 112 427, 2024, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2024.112427.

[46] G. M. van de Ven, T. Tuytelaars, and A. S. Tolias, "Three types of incremental learning," *Nature Machine Intelligence*, vol. 4, no. 12, Dec. 2022. DOI: 10.1038/s42256-022-00568-3.

[47] Y. He and B. Sick, "Clear: An adaptive continual learning framework for regression tasks," *AI Perspectives*, vol. 3, no. 1, p. 2, 2021, ISSN: 2523-398X. DOI: 10.1186/s42467-021-00009-8.

[48] A. G. Menezes, G. de Moura, C. Alves, and A. C. de Carvalho, "Continual object detection: A review of definitions, strategies, and challenges," *Neural Networks*,

vol. 161, pp. 476–493, 2023, ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2023.01.041.

[49] F. Pasti, M. Ceccon, D. D. Pezze, *et al.*, *Latent distillation for continual object detection at the edge*, 2024. arXiv: 2409.01872 [cs.CV].

[50] D. Rao, F. Visin, A. A. Rusu, Y. W. Teh, R. Pascanu, and R. Hadsell, *Continual unsupervised representation learning*, 2019. arXiv: 1910.14481 [cs.LG].

[51] P. Bellavista, S. Dahdal, L. Foschini, D. Tazzioli, M. Tortonesi, and R. Venanzi, "Kubernetes enhanced stateful service migration for ml-driven applications in industry 4.0 scenarios," in *2024 IEEE Annual Congress on Artificial Intelligence of Things (AIoT)*, 2024, pp. 25–31. DOI: 10.1109/AIoT63253.2024.00015.

[52] L. Colombi, I. Boleac, M. Brina, *et al.*, "Multi-cluster mlops platform for industry 5.0," in *2025 IEEE Symposium on Computers and Communications (ISCC)*, IEEE, 2025, pp. 1–6. DOI: accepted\to\be\published.

[53] H. Taherdoost and M. Madanchian, "Multi-criteria decision making (mcdm) methods and concepts," *Encyclopedia*, vol. 3, no. 1, pp. 77–87, 2023. DOI: 10.3390/encyclopedia3010006.

[54] S. K. Sahoo and S. Goswami, "A comprehensive review of multiple criteria decision-making (mcdm) methods: Advancements, applications, and future directions," *Decision Making Advances*, vol. 1, pp. 25–48, Dec. 2023. DOI: 10.31181/dma1120237.

[55] X. Li, L. Pan, and S. Liu, "A survey of resource provisioning problem in cloud brokers," *Journal of Network and Computer Applications*, vol. 203, p. 103 384, 2022, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2022.103384.

[56] R. K. Gavade, "Multi-criteria decision making : An overview of different selection problems and methods," in *International Journal of Computer Science and Information Technologies*, 2014.

[57] G. Odu, "Weighting methods for multi-criteria decision making technique," *Journal of Applied Sciences and Environmental Management*, vol. 23, p. 1449, Sep. 2019. DOI: 10.4314/jasem.v23i8.7.

[58] A. Habbal, S. I. Goudar, and S. Hassan, "A context-aware radio access technology selection mechanism in 5g mobile network for smart city applications," *Journal of Network and Computer Applications*, vol. 135, pp. 97–107, 2019, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2019.02.019.

[59] J. Rezaei, "Best-worst multi-criteria decision-making method," *Omega*, vol. 53, pp. 49–57, 2015, ISSN: 0305-0483. DOI: https://doi.org/10.1016/j.omega.2014.11.009.

[60] N. H. Zardari, K. Ahmed, S. Shirazi, and Z. Yusop, *Weighting Methods and their Effects on Multi-Criteria Decision Making Model Outcomes in Water Resources Management*. Springer, Jan. 2015, ISBN: 978-3-319-12585-5. DOI: 10.1007/978-3-319-12586-2.

[61] N. Azhar, N. A. Mohamed Radzi, and W. S. H. M. Wan Ahmad, "Multi-criteria decision making: A systematic review," *Recent Advances in Electrical & Electronic Engineering (Formerly Recent Patents on Electrical & Electronic Engineering)*, vol. 14, Oct. 2021. DOI: 10.2174/2352096514666211029112443.

[62] M. Ashour and A. Mahdiyar, "A comprehensive state-of-the-art survey on the recent modified and hybrid analytic hierarchy process approaches," *Applied Soft Computing*, vol. 150, p. 111 014, 2024, ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2023.111014.

[63] R. Venanzi, S. Dahdal, M. Solimando, *et al.*, "Enabling adaptive analytics at the edge with the bi-rex big data platform," *Computers in Industry*, vol. 147, p. 103 876, 2023, ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2023.103876.

[64] L. Colombi, S. Dahdal, E. Di Caro, *et al.*, "Efficient data dissemination via semantic filtering at the tactical edge," in *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*, 2024, pp. 457–462. DOI: 10.1109/MILCOM61039.2024.10773700.

[65] R. Fronteddu, U. Ardinghi, L. Colombi, *et al.*, "Semantic information management systems," in *2025 International Conference on Military Communication and Information Systems (ICMCIS)*, 2025, pp. 1–9. DOI: 10.1109/ICMCIS64378.2025.11047741.

[66] S. Goswami, S. Chakraborty, S. Ghosh, A. Chakrabarti, and B. Chakraborty, "A review on application of data mining techniques to combat natural disasters," *Ain Shams Engineering Journal*, vol. 9, no. 3, pp. 365–378, 2018, ISSN: 2090-4479. DOI: https://doi.org/10.1016/j.asej.2016.01.012.

[67] K. Bänsch, J. Busse, F. Meisel, *et al.*, "Energy-aware decision support models in production environments: A systematic literature review," *Computers & Industrial Engineering*, vol. 159, p. 107 456, 2021, ISSN: 0360-8352. DOI: https://doi.org/10.1016/j.cie.2021.107456.

[68] D. Ienco and R. Interdonato, "Deep multivariate time series embedding clustering via attentive-gated autoencoder," in *Advances in Knowledge Discovery and Data Mining*, H. W. Lauw, R. C.-W. Wong, A. Ntoulas, E.-P. Lim, S.-K. Ng, and S. J. Pan, Eds., Cham: Springer International Publishing, 2020, pp. 318–329, ISBN: 978-3-030-47426-3.

[69] V. Peñaloza, "Time2vec embedding on a seq2seq bi-directional lstm network for pedestrian trajectory prediction," *Res. Comput. Sci.*, vol. 149, pp. 249–260, 2020.

[70] S. M. Kazemi, R. Goel, S. Eghbali, *et al.*, "Time2vec: Learning a vector representation of time," *CoRR*, 2019. arXiv: 1907.05321.

[71] N. Seliya, A. Abdollah Zadeh, and T. Khoshgoftaar, "A literature review on one-class classification and its potential applications in big data," *Journal of Big Data*, vol. 8, no. 1, p. 122, 2021. DOI: 10.1186/s40537-021-00514-x.

[72] S. S. Khan and M. G. Madden, "One-class classification: Taxonomy of study and review of techniques," *The Knowledge Engineering Review*, vol. 29, no. 3, pp. 345–374, 2014. DOI: 10.1017/S026988891300043X.

[73] J. Miao, H. Tao, H. Xie, J. Sun, and J. Cao, "Reconstruction-based anomaly detection for multivariate time series using contrastive generative adversarial networks," *Information Processing & Management*, vol. 61, no. 1, p. 103 569, 2024, ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2023.103569.

[74] L. Colombi, N. Belletti, L. Ferrari, *et al.*, "Exploring explainable and causal ai for feature selection in industry 5.0," in *2025 IEEE 30th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2025, pp. 1–6. DOI: 10.1109/CAMAD67323.2025.11229911.

[75] A. Corradi, L. Foschini, C. Giannelli, *et al.*, "Smart Appliances and RAMI 4.0: Management and Servitization of Ice Cream Machines," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 1007–1016, 2019. DOI: 10.1109/TII.2018.2867643.

[76] A. M. Ghosh and K. Grolinger, "Edge-cloud computing for internet of things data analytics: Embedding intelligence in the edge with deep learning," *IEEE*

Transactions on Industrial Informatics, vol. 17, no. 3, pp. 2191–2200, 2021. DOI: 10.1109/TII.2020.3008711.

[77] C. Savaglio, P. Mazzei, and G. Fortino, "Edge Intelligence for Industrial IoT: Opportunities and Limitations," in *Proceedings of 5th International Conference on Industry 4.0 and Smart Manufacturing (ISM 2023)*, 2024, pp. 397–405. DOI: 10.1016/j.procs.2024.01.039.

[78] F. Foukalas and A. Tziouvaras, "Edge artificial intelligence for industrial internet of things applications: An industrial edge intelligence solution," *IEEE Industrial Electronics Magazine*, vol. 15, no. 2, pp. 28–36, 2021.

[79] F. Tabanelli, S. Dahdal, N. Belletti, *et al.*, "Smart and sustainable ice cream making through edge machine learning," *IEEE Transactions on Industrial Informatics*, pp. 1–10, 2026. DOI: 10.1109/TII.2025.3649155.

[80] S. Shao, S. McAleer, R. Yan, and P. Baldi, "Highly accurate machine fault diagnosis using deep transfer learning," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2446–2455, 2019. DOI: 10.1109/TII.2018.2864759.

[81] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," in *Proceedings of the European Conference on Computer Vision (ECCV 2014)*, 2014, pp. 346–361. DOI: 10.1007/978-3-319-10578-923.

[82] M. A. Hossain, S. K. Ray, and J. Lota, "Smartdr:a device-to-device communication for post-disaster recovery," *Journal of Network and Computer Applications*, vol. 171, p. 102 813, 2020, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2020.102813.

[83] S. Abdellatif, O. Tibermacine, W. Bechkit, and A. Bachir, "Heterogeneous iot/lte prose virtual infrastructure for disaster situations," *Journal of Network and Com-*

puter Applications, vol. 213, p. 103 602, 2023, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2023.103602.

[84] S. Dahdal, S. Cavicchi, A. Gilli, *et al.*, "Roamml distributed continual learning: Adaptive and flexible data-driven response for disaster recovery operations," *Journal of Network and Computer Applications*, p. 104 322, 2025, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2025.104322.

[85] S. Dahdal, A. Gilli, F. Poltronieri, M. Tortonesi, C. Stefanelli, and N. Suri, "Roamml platform: Enabling distributed continual learning for disaster relief operations," in *2024 IEEE Symposium on Computers and Communications (ISCC)*, 2024, pp. 1–6. DOI: 10.1109/ISCC61673.2024.10733586.

[86] S. Dahdal, F. Poltronieri, A. Gilli, *et al.*, "Roamml: Distributed machine learning at the tactical edge," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, 2023, pp. 33–38. DOI: 10.1109/MILCOM58377.2023.10356274.

[87] P. Buzzega, M. Boschini, A. Porrello, and S. Calderara, "Rethinking experience replay: A bag of tricks for continual learning," *CoRR*, vol. abs/2010.05595, 2020. arXiv: 2010.05595.

[88] D. McCrory, *Data gravity – in the clouds*, en, Dec. 2010.

[89] S. Tavakkol, H. To, S. H. Kim, P. Lynett, and C. Shahabi, "An entropy-based framework for efficient post-disaster assessment based on crowdsourced data," *Proceedings of the Second ACM SIGSPATIALInternational Workshop on the Use of GIS in Emergency Management*, pp. 1–8, Oct. 2016. DOI: 10.1145/3017611.3017624.

[90] S. Chibani and F.-X. Coudert, "Machine learning approaches for the prediction of materials properties," *APL Materials*, vol. 8, no. 8, p. 080 701, Aug. 2020, ISSN: 2166-532X. DOI: https://doi.org/10.1063/5.0018384.

[91] L. Budach, M. Feuerpfeil, N. Ihde, *et al.*, *The effects of data quality on machine learning performance*, 2022. arXiv: 2207.14529 [cs.DB].

[92] N. Suri, G. Benincasa, R. Lenzi, M. Tortonesi, C. Stefanelli, and L. Sadler, "Exploring value-of-information-based approaches to support effective communications in tactical networks," *IEEE Communications Magazine*, vol. 53, no. 10, pp. 39–45, 2015. DOI: 10.1109/MCOM.2015.7295461.

[93] A. Garcia-Perez, R. Miñón, A. I. Torre-Bastida, and E. Zulueta-Guerrero, "Analysing edge computing devices for the deployment of embedded ai," *Sensors*, vol. 23, no. 23, 2023, ISSN: 1424-8220. DOI: 10.3390/s23239495.

[94] P. K. D. Pramanik, S. Biswas, S. Pal, D. Marinković, and P. Choudhury, "A comparative analysis of multi-criteria decision-making methods for resource selection in mobile crowd computing," *Symmetry*, vol. 13, no. 9, 2021, ISSN: 2073-8994. DOI: 10.3390/sym13091713.

[95] M. S. Aslanpour, S. S. Gill, and A. N. Toosi, "Performance evaluation metrics for cloud, fog and edge computing: A review, taxonomy, benchmarks and standards for future research," *Internet of Things*, vol. 12, p. 100 273, 2020, ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2020.100273.

[96] A. S. Haichour and K. Benfriha, "Empowering real-time iot applications: A brief review on leveraging gpu acceleration for latency reduction," in *Internet of Things. 7th IFIPIoT 2024 International IFIP WG 5.5 Workshops*, G. Rey, J.-Y. Tigli, and E. Franquet, Eds., Cham: Springer Nature Switzerland, 2025, pp. 107–120, ISBN: 978-3-031-82065-6.

[97] A. Haroon, M. Sagor, M. Maurice, L. Jin, R. Stoleru, and R. Blalock, "On edge coordination in highly dynamic cyber-physical systems for emergency response," in *2022 Workshop on Cyber Physical Systems for Emergency Response (CPS-ER)*, 2022, pp. 7–12. DOI: 10.1109/CPS-ER56134.2022.00008.

[98] A. Chaudhry, P. K. Dokania, T. Ajanthan, and P. H. S. Torr, "Riemannian walk for incremental learning: Understanding forgetting and intransigence," in *Lecture Notes in Computer Science*. Springer International Publishing, 2018, pp. 556–572, ISBN: 9783030012526. DOI: 10.1007/978-3-030-01252-6_33.

[99] G. M. van de Ven and A. S. Tolias, *Three scenarios for continual learning*, 2019. arXiv: 1904.07734 [cs.LG].

[100] M. Viswanathan, *Wireless Communication Systems in Matlab*. Independently published, 2020, ISBN: 979-8648350779.

[101] M. Aruldoss, T. Lakshmi, and V. Venkatesan, "A study on evaluation metrics for multi criteria decision making (mcdm) methods - topsis, copras & gra," *International Journal of Computing Algorithm*, vol. 7, pp. 29–37, Jun. 2018. DOI: 10.20894/IJCOA.101.007.001.006.

[102] A. E. Youssef, "An integrated mcdm approach for cloud service selection based on topsis and bwm," *IEEE Access*, vol. 8, pp. 71 851–71 865, 2020. DOI: 10.1109/ACCESS.2020.2987111.

[103] R. Venanzi, L. Colombi, D. Tazzioli, S. Dahdal, M. Tortonesi, and L. Foschini, "Collective intelligence-based service migration enabling zoom-in functionality within industry 5.0," *Internet of Things*, 2025. DOI: https://doi.org/10.1016/j.iot.2025.101830.

[104] F. TASSI, R. Lazzarini, E. BELLODI, S. DAHDAL, C. Stefanelli, and M. TORTONESI, *Machine for making liquid, semiliquid or semisolid food products and related control method*, US Patent App. 18/768,782, Jan. 2025.

[105] X. Mi, M. Tang, H. Liao, W. Shen, and B. Lev, "The state-of-the-art survey on integrations and applications of the best worst method in decision making: Why, what, what for and what's next?" *Omega*, vol. 87, pp. 205–225, 2019, ISSN: 0305-0483. DOI: https://doi.org/10.1016/j.omega.2019.01.009.

[106] F. A. Potra and S. J. Wright, "Interior-point methods," *Journal of Computational and Applied Mathematics*, vol. 124, no. 1, pp. 281–302, 2000, Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations, ISSN: 0377-0427. DOI: https://doi.org/10.1016/S0377-0427(00)00433-7.

[107] Hamdani and R. Wardoyo, "The complexity calculation for group decision making using topsis algorithm," *AIP Conference Proceedings*, vol. 1755, no. 1, p. 070 007, Jul. 2016, ISSN: 0094-243X. DOI: 10.1063/1.4958502.

Author's Publications

[1] F. TASSI, R. Lazzarini, E. BELLODI, S. DAHDAL, C. Stefanelli, and M. TORTONESI, *Machine for making liquid, semiliquid or semisolid food products and related control method*, US Patent App. 18/768,782, Jan. 2025. DOI: https://patents.google.com/patent/US20250024852A1/en.

[2] S. Dahdal, S. Cavicchi, A. Gilli, *et al.*, "Roamml distributed continual learning: Adaptive and flexible data-driven response for disaster recovery operations," *Journal of Network and Computer Applications*, p. 104 322, 2025, ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2025.104322.

[3] R. Venanzi, S. Dahdal, M. Solimando, *et al.*, "Enabling adaptive analytics at the edge with the bi-rex big data platform," *Computers in Industry*, vol. 147, p. 103 876, 2023, ISSN: 0166-3615. DOI: https://doi.org/10.1016/j.compind.2023.103876.

[4] L. Colombi, M. Vespa, N. Belletti, *et al.*, "Embedding models for multivariate time series anomaly detection in industry 5.0," *Data Science and Engineering*, Jun. 2025, ISSN: 2364-1541. DOI: 10.1007/s41019-025-00295-w.

[5] S. Dahdal, L. Colombi, M. Brina, *et al.*, "An mlops framework for gan-based fault detection in bonfiglioli's evo plant," *Infocommunications journal*, vol. 16, no. 2, pp. 2–10, 2024, ISSN: 2061-2079. DOI: 10.36244/icj.2024.2.1.

[6] R. Venanzi, L. Colombi, D. Tazzioli, S. Dahdal, M. Tortonesi, and L. Foschini, "Collective intelligence-based service migration enabling zoom-in functionality within industry 5.0," *Internet of Things*, 2025. DOI: https://doi.org/10.1016/j.iot.2025.101830.

[7] F. Tabanelli, S. Dahdal, N. Belletti, *et al.*, "Smart and sustainable ice cream making through edge machine learning," *IEEE Transactions on Industrial Informatics*, pp. 1–10, 2026. DOI: 10.1109/TII.2025.3649155.

[8] S. Dahdal, F. Poltronieri, M. Tortonesi, C. Stefanelli, and N. Suri, "A data mesh approach for enabling data-centric applications at the tactical edge," in *2023 International Conference on Military Communications and Information Systems (ICMCIS)*, 2023, pp. 1–9. DOI: 10.1109/ICMCIS59922.2023.10253568.

[9] S. Dahdal, F. Poltronieri, A. Gilli, *et al.*, "Roamml: Distributed machine learning at the tactical edge," in *MILCOM 2023 - 2023 IEEE Military Communications Conference (MILCOM)*, 2023, pp. 33–38. DOI: 10.1109/MILCOM58377.2023.10356274.

[10] S. Dahdal and M. Tortonesi, "Enabling big data and machine learning applications in high-stakes environments," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, IEEE, May 2024, pp. 1–4. DOI: 10.1109/noms59830.2024.10574906.

[11] L. Colombi, A. Gilli, S. Dahdal, *et al.*, "A machine learning operations platform for streamlined model serving in industry 5.0," in *NOMS 2024-2024 IEEE Network Operations and Management Symposium*, 2024, pp. 1–6. DOI: 10.1109/NOMS59830.2024.10575103.

[12] P. Bellavista, S. Dahdal, L. Foschini, D. Tazzioli, M. Tortonesi, and R. Venanzi, "Kubernetes enhanced stateful service migration for ml-driven applications in industry 4.0 scenarios," in *2024 IEEE Annual Congress on Artificial Intelligence of Things (AIoT)*, 2024, pp. 25–31. DOI: 10.1109/AIoT63253.2024.00015.

[13] S. Dahdal, A. Gilli, F. Poltronieri, M. Tortonesi, C. Stefanelli, and N. Suri, "Roamml platform: Enabling distributed continual learning for disaster relief operations," in *2024 IEEE Symposium on Computers and Communications (ISCC)*, 2024, pp. 1–6. DOI: 10.1109/ISCC61673.2024.10733586.

[14] L. Colombi, S. Dahdal, E. Di Caro, *et al.*, "Efficient data dissemination via semantic filtering at the tactical edge," in *MILCOM 2024 - 2024 IEEE Military Communications Conference (MILCOM)*, 2024, pp. 457–462. DOI: 10.1109/MILCOM61039.2024.10773700.

[15] L. Colombi, M. Vespa, N. Belletti, *et al.*, *Multivariate time series anomaly detection in industry 5.0*, 2025. arXiv: 2503.15946 [cs.LG].

[16] L. Colombi, M. Brina, M. Vespa, *et al.*, "Optimizing industry 5.0 machine learning-based applications via synthetic data generation," in *2024 IEEE 29th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2024, pp. 1–6. DOI: 10.1109/CAMAD62243.2024.10942898.

[17] R. Fronteddu, U. Ardinghi, L. Colombi, *et al.*, "Semantic information management systems," in *2025 International Conference on Military Communication and Information Systems (ICMCIS)*, 2025, pp. 1–9. DOI: 10.1109/ICMCIS64378.2025.11047741.

[18] L. Colombi, E. D. Caro, S. Dahdal, *et al.*, "Fededge-learn: A semi-supervised federated learning framework for industry 5.0," in *2025 IEEE Symposium on*

Computers and Communications (ISCC), 2025, pp. 1–6. DOI: 10.1109/ISCC65549.2025.11326181.

[19] L. Colombi, I. Boleac, M. Brina, *et al.*, "Multi-cluster mlops platform for industry 5.0," in *2025 IEEE Symposium on Computers and Communications (ISCC)*, 2025, pp. 1–6. DOI: 10.1109/ISCC65549.2025.11325976.

[20] L. Colombi, N. Belletti, L. Ferrari, *et al.*, "Exploring explainable and causal ai for feature selection in industry 5.0," in *2025 IEEE 30th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, 2025, pp. 1–6. DOI: 10.1109/CAMAD67323.2025.11229911.

List of Figures

List of Tables