UNIVERSITÀ DEGLI STUDI DI PALERMO

PhD course in Information and Communication Technologies
Department of Engineering

Shaping a Sustainable Future with IoT and AI for Water Distribution Networks

AUTHOR
Ing. Gabriele Restuccia

COORDINATOR
Prof. Ilenia Tinnirello

TUTOR
Prof. Ilenia Tinnirello

CO-TUTOR
Prof. Domenico Garlisi

XXXVII Cycle
ACADEMIC YEAR 2023 - 2024

Acknowledgements

This incredibly long journey would not have been possible without the most important people in my life: my family. I owe thanks to my mother and father for their endless patience, support, and love, and to my brother and sister for always understanding and caring for me. Can I forget Bruno again, like last time? Of course not, but the temptation to leave him out and spark more laughter was almost irresistible!

Staying on the family theme, over the past few years the presence of Rosaria has been the most precious and beautiful thing that could have happened to me. I will never be able to thank her enough except by giving her more than all the possible love and reciprocating all the support and understanding she has given me.

Throughout this time, the presence of my dearest friends, Fabrizio and Claudia, has been one of the few constants, and I cannot thank them enough either.

I am also grateful to all the wonderful colleagues I met along the way. Domenico, Daniele, and Fabrizio were wise mentors. Antonino, Alessandra, Lino, Gloria, Farzam, the newly arrived Silvia, and everyone else who passed through our lab have been great companions and, above all, friends. Last but certainly not least, I especially thank Ilenia for her splendid guidance, for waiting, supporting, understanding, and motivating me to always become a better researcher. I will never forget it.

An affectionate thought also goes to my American colleagues, with whom I shared a brief but intense experience. Federico was an unforgettable roommate. Francesco was a constant spark and inspiration, completely changing my perspective and making me feel at home on another continent.

In my last thesis, I also thanked Cesar, saying that although he couldn't read, he already knew everything there was to know. I like to think that it's still the same today.

Abstract

This thesis addresses both environmental and technical challenges in developing sustainable systems for critical infrastructures, with a particular focus on Water Distribution Networks (WDNs). By leveraging Internet of Things (IoT), artificial intelligence (AI), particularly Machine Learning (ML), and Low Power Wide Area Networks (LPWANs) technologies, specifically Long Range Wide Area Network (LoRaWAN), this research aims to optimize and enhance the resilience of WDNs amidst rising global water demands and climate change. With a projected 40% global water deficit by 2030, improving the efficiency of WDNs is essential for sustainable water management. Additionally, the increasing complexity of modern WDNs, driven by urbanization, fluctuating consumer demands, and limited resources, makes their management more challenging. Beyond these environmental and managerial concerns, this work also addresses the security vulnerabilities associated with the rapid expansion of these interconnected systems, which are often constrained by limited computational resources and minimal security, posing significant risks to critical infrastructure.

The first part of the thesis explores IoT-based solutions for improving sustainability in WDNs.

Chapter 1 introduces the key concepts in IoT, ML, adversarial machine learning (AML), and WDNs, laying the foundation for the following research.

Chapter 2 emphasizes the integration of IoT, edge computing, and ML, demonstrating how computations at the network edge can optimize performance. Strategic placement of LoRaWAN gateways is identified as key to maximizing data quality and system efficiency. Additionally, multiple ML models, including Multilayer Perceptrons (MLPs), convolutional neural networks (CNNs), and Graph Neural Networks (GNNs), are evaluated for predicting hydraulic values and highlighting network anomalies, such as random leakages, under varying network conditions.

Chapter 3 introduces SWIM, a comprehensive platform that integrates IoT, Digital Twins (DTWs), simulations, and ML to streamline the management of WDNs. SWIM enables real-time monitoring and data-driven decision-making, leveraging simulations and DTWs to address the complexity of modern water distribution systems.

The second part of the thesis shifts focus to security, particularly in IoT authentication and ML security.

Chapter 4 introduces RIOT-AKA, a novel authentication framework adapted from cellular networks, providing a robust yet efficient solution for securing large-scale IoT environ-

ments with constrained computational resources.

Chapter 5 explores AML vulnerabilities in MLs-based systems. The Subgraph Extraction-based Attack (SSEA) is proposed, a method that exploits semantic relationships between classes in CNNs to reduce computational complexity while maintaining high attack efficiency. This technique highlights the increasing need for ML security in critical infrastructures.

The thesis concludes with Chapter 6 summarizing its contributions.

Contents

List of Figures

List of Tables

Derived Works

This thesis builds upon the following published and ongoing works:

- G. Bianchi, A. La Rosa, **G. Restuccia**, ”RIOT-AKA: cellular-like authentication over IoT devices,” 2021 IEEE 29th International Conference on Network Protocols (ICNP), pp. 1-6, IEEE, 2021.
- D. Garlisi, **G. Restuccia**, I. Tinnirello, F. Cuomo, I. Chatzigiannakis, ”Leakage detection via edge processing in LoRaWAN-based smart water distribution networks,” 2022 18th International Conference on Mobility, Sensing and Networking (MSN), pp. 223-230, IEEE, 2022.
- **G. Restuccia**, I. Tinnirello, F. Lo Valvo, G. Baiamonte, D. Garlisi, C. Giaconia, ”A distributed analysis of vibration signals for leakage detection in Water Distribution Networks,” Materials Research Proceedings, vol. 26, Materials Research Forum LLC, 2023.
- D. Garlisi, **G. Restuccia**, I. Tinnirello, F. Cuomo, I. Chatzigiannakis, ”Real-Time Leakage Zone Detection in Water Distribution Networks: A Machine Learning-Based Stream Processing Algorithm,” International Symposium on Algorithmic Aspects of Cloud Computing, pp. 86-99, Springer Nature Switzerland, 2023.
- **G. Restuccia**, I. Tinnirello, F. Restuccia, ”Semantic Subgraph Extraction Attacks To Convolutional Neural Networks,” ongoing research, University of Palermo, Northeastern University.
- **G. Restuccia**, I. Tinnirello, F. Restuccia, ”Enhancing Adversarial Attacks on Wireless Communication Systems: A Low-SNR Targeted Approach,” ongoing research, University of Palermo, Northeastern University.
- **G. Restuccia**, D. Garlisi, F. Giuliano, ”A Digital Twin Framework for Smart Water Distribution Networks Using Data-Driven Models” ongoing research, University of Palermo (author list subject to expansion).

Chapter 1

Foundational Concepts in IoT, ML, and Security for Critical Infrastructures

1.1 Introduction

This chapter lays the foundation for the research by presenting the key concepts of Internet of Things (IoT) implementations, Machine Learning (ML) theory and applications, security challenges, and the characteristics and tools associated with Water Distribution Networks (WDNs). It prepares for an in-depth exploration of the technologies enabling sustainable and secure modern systems.

1.2 Internet of Things

IoT is a transformative technology that connects billions of physical devices to the internet, fundamentally reshaping telecommunications and various industries. This interconnected network includes a wide array of intelligent devices, such as sensors, actuators, and smartphones, enabling them to communicate and perform tasks that enhance efficiency and simplify daily life. Applications of IoT span across diverse sectors, including smart homes, cities, agriculture, healthcare, and retail, allowing for advanced services like automation, real-time monitoring, and data-driven decision-making [1].

The rapid growth of IoT has led to a highly interconnected world, where devices seamlessly communicate within local networks and across broader systems, leveraging standard communication protocols to deliver innovative services to users [2]. This evolution mirrors the broader development of the internet, which has expanded from simple computer-to-computer connections to a vast web linking billions of smart devices. Since 2008, the number of connected IoT devices has exceeded the global population, with forecasts predicting this figure could reach 80 billion by 2025 [3].

As the internet continues to evolve, a significant shift is occurring from traditional human-to-human communication to Machine-to-Machine (M2M) communication, where devices interact autonomously to exchange information and perform tasks [4]. This shift is expected

to greatly impact how data is processed and utilized, driving new efficiencies and capabilities in both personal and industrial contexts.

The concept of the Internet of Things was first introduced by Kevin Ashton in 1999 [5], and since then, IoT has been anticipated to provide ubiquitous access to extensive data about devices, environments, and processes, fundamentally changing data accessibility and usage. Historically, industries relied heavily on human oversight and manual processes; however, with IoT, many sectors now leverage automated, sensor-driven monitoring and control systems, drastically reducing the need for human intervention and increasing operational efficiency.

The adoption of IoT has accelerated across various industries, becoming essential for improving information sharing and operational processes within and between sectors [6]. Its applications have broadened significantly, from home automation to industrial IoT, enabling the networked communication of physical objects in virtually any location [5]. This expansion is reflected in the growing number of IoT-related projects in fields such as agriculture, security surveillance, environmental monitoring, and food processing, among others. Additionally, the volume of research and development in IoT continues to surge, reflecting the broadening scope and potential of this technology.

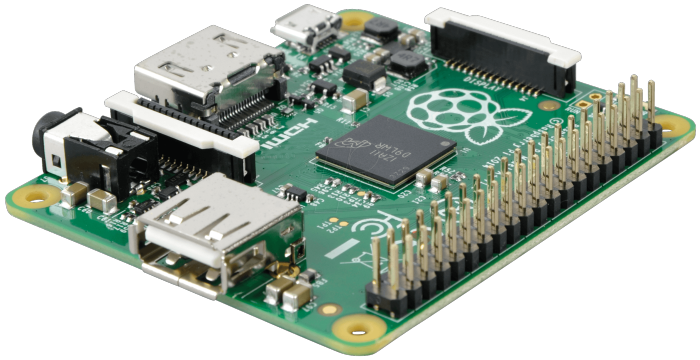


Figure 1.1: The Raspberry Pi is a famous IoT device in the consumer world.

1.2.1 IoT Communication Protocols

The integration of wireless sensor networks (WSNs) and related technologies into IoT has significantly transformed various industries by facilitating the continuous collection and analysis of vast amounts of data. WSNs, along with wireless sensor and actuator networks (WSANs), virtual sensor networks (VSNs), and other devices such as robots and machines, are pivotal in applications ranging from industrial control and health monitoring to traffic management and environmental surveillance [7, 8, 9, 10, 5].

A key enabler of large-scale IoT deployment is Low Power Wide Area Network (LPWAN) technology, which addresses the need for extensive geographic coverage, low power consumption, and minimal data rates in IoT systems. LPWAN is particularly effective for devices distributed across large areas, offering reliable connectivity over distances of several kilometers, thus ensuring that even remote devices remain connected [11]. This technology

prioritizes energy efficiency, allowing IoT devices to operate on battery power for extended periods—often up to a decade—without frequent maintenance or battery replacements, which is especially valuable in large-scale deployments where manual battery management would be impractical and expensive [11].

LPWAN supports several communication protocols specifically tailored for IoT applications, including LoRaWAN, Sigfox, and Narrowband Internet of Things (NB-IoT). These protocols are designed to transmit small data packets at infrequent intervals, matching the typical usage patterns of IoT devices that primarily relay status updates or sensor readings [12]. Among these, LoRaWAN has gained prominence due to its ability to transmit data over long distances with high reliability and broad coverage. Developed by Semtech, LoRaWAN operates at the physical layer (PHY) with data transfer rates ranging from 250 bps to 5.5 kbps and bandwidths between 125 and 500 kHz, supporting network architectures that enable thousands of nodes to communicate with gateways simultaneously [13, 14].

The integration of LPWANs technologies into IoT networks is critical for enhancing the reach, scalability, and sustainability of IoT systems, especially in scenarios requiring long-range communication and efficient energy use. By leveraging LPWANs, IoT applications can achieve new levels of durability and scalability, making it an essential component in the deployment of modern smart systems [11, 15, 12].

Other communication protocols such as CoAP, MQTT, and Apache Kafka further enable seamless data exchange in IoT networks, each designed for different communication requirements:

- CoAP (Constrained Application Protocol):
  - Optimized for constrained devices and low-bandwidth networks.
  - Operates over UDP, using a request/response model similar to HTTP but with significantly lower overhead.
  - Designed for energy-efficient M2M communication, particularly in environments where devices have limited processing power [16, 17].
- MQTT (Message Queuing Telemetry Transport):
  - Lightweight publish/subscribe protocol suited for low-bandwidth, high-latency networks.
  - Operates over TCP, ensuring reliable message delivery.
  - Ideal for real-time IoT applications where stable, continuous communication is critical, such as in industrial automation and health monitoring [18, 19].
- Apache Kafka:
  - High-throughput, distributed messaging system designed for large-scale, event-driven architectures.

- Operates with a publish/subscribe model and supports message partitioning for scalability.
- Stores messages on disk, allowing consumers to access historical data and ensuring persistence and reliability, but with higher resource demands compared to MQTT and CoAP [20, 21].

1.2.2 IoT Advantages

IoT offers several key advantages that drive its adoption across various sectors. One of the most significant benefits is the technological advancement it facilitates, enhancing the functionality and performance of connected devices through real-time data measurement and analysis. This capability provides a precise reflection of real-world conditions, enabling more accurate and responsive operations [5]. Another critical advantage is the improved consumer engagement that IoT enables. Traditional methods often struggle with ambiguity and inaccuracies, especially in the context of the Industrial Internet of Things (Industrial Internet of Things (IIoT)), where clear and precise interactions are crucial. IoT addresses these challenges by fostering more interactive and meaningful engagements, thereby enhancing the overall user experience [5]. Furthermore, IoT revolutionizes data collection processes by overcoming the limitations of traditional methods, which are often constrained by practicality. By integrating data into accessible and actionable formats, IoT allows for more comprehensive analysis and a deeper understanding of various phenomena, leading to innovative approaches in data utilization [5, 22, 23]. Finally, IoT significantly boosts resource efficiency by providing real-time insights that enable better management and optimization of resources. This transition from abstract statistical data to concrete, actionable insights reduces waste and improves overall operational efficiency, allowing for more strategic allocation and use of resources [5].

1.2.3 IoT Challenges

Despite its numerous advantages, IoT also faces several critical challenges, particularly in the areas of security, privacy, complexity, and integration. A primary concern is the vulnerability to cyber-attacks due to the inherent interconnectivity of IoTs devices. This interconnected nature can lead to the exposure of personal data, even when users are not actively involved, raising significant concerns about the security of data during storage and transmission as the volume of IoT data continues to grow [5]. The complexity of implementing IoT systems is another major hurdle. The development, design, and maintenance of IoT technologies require meticulous planning and execution due to their intricate and multifaceted nature, which complicates their deployment and ongoing management [5]. Flexibility and integration pose additional challenges. Consumers often express concerns about the compatibility and ease of integration of IoTs devices, especially when dealing with conflicting or proprietary source codes. These issues can impede the seamless adoption and broader integration of IoT tech-

nologies [5]. Finally, IoT also complicates marketing and content delivery. Although IoT enhances data collection by providing insights into user behaviors, the sheer volume of data can make it challenging to effectively manage and leverage these insights for content delivery and marketing strategies. This underscores the critical need for efficient data management and analysis within IoT systems to fully realize their potential [22].

1.2.4 IoT Architectural Layers

The architecture of IoT is composed of several layers, each serving a distinct function within the system [5]:

- **Sensing Layer:** Also referred to as the perception layer, this layer is responsible for incorporating physical objects and capturing data through technologies such as Radio Frequency Identification (RFID) and sensors [24]. This layer significantly enhances the IoT's ability to detect and identify objects or environments by employing advanced technological methods. Each device or service within this layer is assigned a universally unique identifier (UUID), which is essential for specific industry sectors [25]. Devices in this layer become "smart" or "intelligent" due to their capability to interact with the external environment and perform functions such as self-identification, self-diagnosis, and self-testing [26, 27, 28, 5].

- **Networking Layer:** The networking layer facilitates the transfer of data between the sensing layer and the processing layer by providing networking support across both wireless and wired networks. This layer collects data from existing IT infrastructures such as business systems, transportation systems, power grids, healthcare systems, and Information and Communication Technology (ICT) systems [25]. It encompasses the technologies and protocols that enable these connections and is not to be confused with the network layer in the ISO/OSI model, which focuses solely on data routing within the network [29, 27]. Wireless protocols are especially critical in this layer, allowing for flexible deployment of sensors in hard-to-reach areas and enabling easy addition or removal of nodes without restructuring the entire network. The choice of protocol depends on factors like network size, node power consumption, and required transmission speed [26].

- **Service Layer:** Also known as the application layer, this layer is responsible for managing and creating services that fulfill user requirements [25]. To ensure efficient service delivery, this layer incorporates enabling technologies such as resource management, interface technology, service management, and middleware [26]. The service layer supports a wide range of applications, from agriculture to smart healthcare, by processing and storing data through analytical tools like databases and software [29, 28]. Data formats handled in this layer vary, including binary-based data that is tiny and illegible to humans, and larger, text-based data that is human-readable [26]. Mid-

dleware services, a key component of this layer, are essential for meeting common application requirements [25].

- **Interface Layer:** The interface layer provides the necessary services for user interaction and connectivity with other applications. This layer simplifies the management and connectivity of devices through interface profiles (IFPs), which are subsets of service standards that enable interaction with distributed applications on the network [25]. A significant element of this layer is the implementation of Universal Plug and Play (UPnP), which ensures interoperability among services provided by various devices [25, 30, 31].

1.2.5 Green IoT

As IoT continues to expand, the efficiency and energy consumption of connected devices have become critical areas of focus. Each active device, although requiring minimal power individually, collectively demands substantial energy due to the sheer volume of billions of devices. This significant energy requirement necessitates considerable processing capabilities from large data centers, making energy management a crucial issue [32]. WSNs are a vital component of IoT, integrating heterogeneous systems, data, and applications. Sensor nodes within WSNs detect information, perform processing, and communicate with other nodes. However, these nodes are often powered by batteries with limited lifespans, constraining their processing ability, memory, and communication capabilities [33, 34]. Energy efficiency is crucial for WSNs and IoT devices, as most energy is consumed in data processing and transmission [35]. Inefficient scheduling and routing algorithms can lead to unnecessary energy expenditure on protocol overheads, the transmission of redundant data, or retransmissions. Therefore, designing and implementing efficient load balancing schemes and energy gauge nodes is crucial to maximizing the lifespan of these constraint-oriented networks [36]. To address the energy concerns associated with the rapid growth of IoT, the concept of Green Internet of Things (GIoT) has emerged. The primary objective of the GIoT initiative is to make IoT systems more energy-efficient from the design phase through to implementation and enhancing energy efficiency and reducing CO_2 emissions associated with IoT technology [32]. Research on GIoT can be broadly categorized into two main areas [32]: (i) software solutions, which aim to reduce energy waste through more efficient resource utilization, and (ii) hardware improvements, which focus on increasing the energy efficiency of IoT devices. Software solutions include optimizing algorithms, developing energy-efficient communication protocols, and implementing intelligent resource management techniques. Hardware improvements involve designing low-power sensors, energy-efficient processors, and utilizing advanced materials that reduce power consumption. The importance of GIoT extends beyond energy savings. It also aims to mitigate the environmental impact of IoT technology by reducing CO_2 emissions and promoting eco-sustainability. As traditional energy sources become scarcer and energy consumption continues to rise, the need for GIoT

solutions becomes increasingly critical. GIoT not only addresses the energy efficiency of individual devices but also considers the broader impact of IoT systems on the environment.

1.2.6 Cloud and Edge Iot Computing

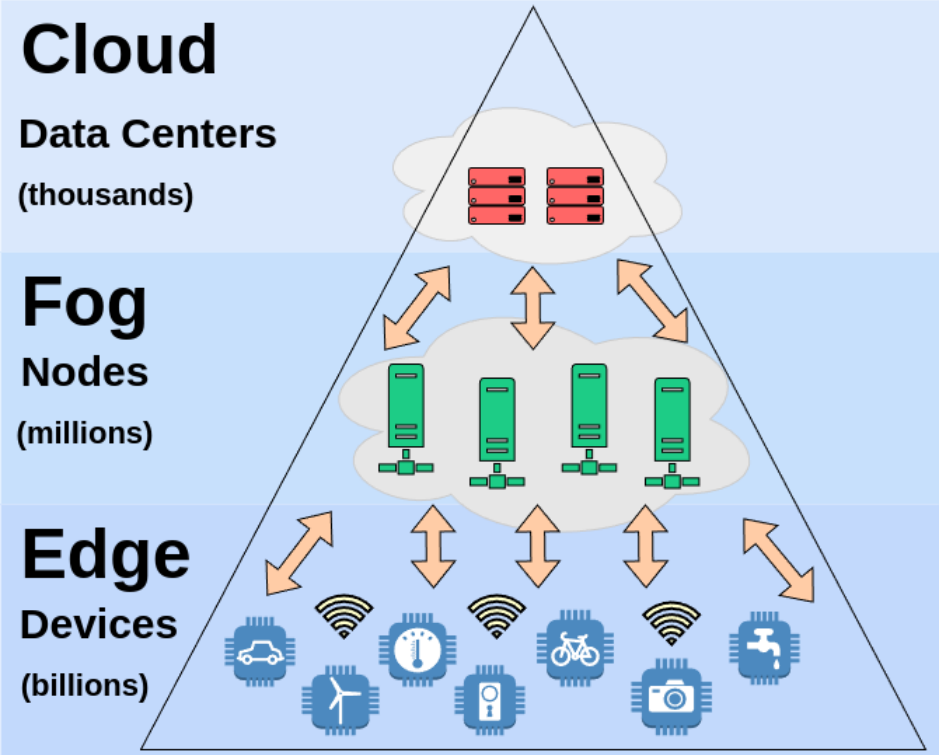


Figure 1.2: Hierarchical architecture of edge, fog, and cloud computing.

While much of IoT data analytics is performed in the cloud, this centralized approach can introduce latency, particularly in time-sensitive applications [37]. This issue becomes even more pronounced as the number of IoT devices increases. For example, in connected vehicle systems, real-time or near real-time decision-making is critical. As the network of connected vehicles grows, ensuring a balance between analysis speed and scalability becomes increasingly important. Figure 1.2 shows the hierarchical architecture of edge, fog, and cloud computing. At the base, billions of edge devices collect and process data locally. In the fog layer, millions of intermediary nodes manage and process data closer to the edge, reducing latency. The cloud layer, consisting of thousands of data centers, provides centralized processing and storage for large-scale data management and long-term analysis.

Cloud Computing

Cloud computing utilizes remote servers via the Internet to store, manage, and process data. In recent years, it has gained significant attention due to its scalable infrastructure and ability to support diverse services such as Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [38]. These services are delivered through the

”everything as a Service” (XaaS) model, which offers flexibility across different IoT applications. Cloud computing is especially advantageous in IoT environments for handling the large volumes of data generated by smart devices. Cloud architectures can be customized based on specific use cases, including public, private, hybrid, or community-based models [38].

Edge Computing

Edge computing addresses the limitations of cloud computing by processing data near its source, reducing reliance on centralized cloud infrastructure [39]. Sending raw IoT data to the cloud can be costly in terms of bandwidth and latency, especially for time-critical applications. Edge computing mitigates these issues by performing local computations within the IoT network itself. This reduces bandwidth consumption, lowers latency, and decreases the risk of a single point of failure. Unlike the centralized servers of cloud computing, edge computing relies on smaller, distributed servers. It is better suited for real-time, low-latency tasks, while cloud computing is more appropriate for complex, delay-tolerant operations [38]. Figure 3 depicts a collaborative edge-cloud model, where data from IoT devices is processed at the edge before being relayed to the core network. It is predicted that approximately 45% of IoT data will rely on edge architectures in the near future [40].

Fog Computing

Fog computing serves as an intermediary between cloud and edge computing, extending the network by offering computation, storage, and networking services closer to the data source [41]. Originally proposed by Cisco [42], fog computing addresses the latency constraints of time-sensitive applications. While edge computing handles data processing at the device level, fog computing operates at the network level, providing additional resources through virtualized platforms. According to Cisco’s whitepaper [43], fog nodes excel in time-sensitive analytics with response times ranging from milliseconds to minutes, although they have limited storage capacity compared to cloud infrastructure, which can retain data for extended periods [43].

Figure 2 outlines the components of IoT ecosystems and their relation to data analytics. While cloud servers are ideal for complex, resource-intensive analyses, lightweight algorithms are more appropriate for edge computing, as shown in Figure 3.

1.3 Water Distribution Networks

WDNs are crucial to modern society, providing the water necessary for daily human activities. Nearly 99% of the global population depends on these systems for their water supply from morning until night [44]. However, WDNs do more than just distribute water through a network of pipelines; they encompass various factors, functions, and associated challenges that require careful management.

Water consumption is typically measured using metering systems, but both traditional and advanced systems present significant challenges. Advanced meters, for instance, often face issues such as high costs, manual recording, meter reliability, limited lifespan, and the need for continuous power supply. Additionally, human involvement in tasks like recording meter readings, billing, and data entry increases the risk of errors and extends the time needed for accurate data collection [44].

Water audits are essential for predicting future water conservation patterns and facilitating strategic planning to ensure sufficient water supply for the future. These audits rely on accurate, real-time data on water consumption, which is instrumental in developing effective water supply strategies.

Another major challenge is the scarcity of government funding for water distribution agencies [44]. Without adequate financial resources, these agencies may struggle to maintain services for all users. One solution to this issue is the implementation of systems that enable faster cost recovery, improving the financial sustainability of WDN operations and their management.

1.3.1 Traditional vs. Modern Approaches

Historically, the management and analysis of Water Distribution Systems (WDSs) has relied on mathematical models and computational simulations that use physical laws of mass and energy conservation, head loss-flow relationships, and pressure requirements to simulate network behavior under various conditions. Tools such as EPANET, Porteau, WaterCAD, WATSYS, and WATNET have been widely adopted for hydraulic analysis and system optimization [11].

However, traditional techniques often struggle with the inherent complexity and nonlinearity of real-world WDSs, particularly in stochastic scenarios. The introduction of artificial intelligence (AI) has created new opportunities to enhance WDS management by utilizing advanced data-driven methods for leak detection, water security, and pressure control. AI algorithms can process large volumes of heterogeneous data, offering predictions and optimizations that surpass the capabilities of classical methods [12].

1.3.2 Smart Water Distribution Networks for Environmental Sustainability

Effective water management is critical not only for human survival but also for environmental sustainability, food production, wastewater treatment, irrigation, and maintaining energy balance [45]. One of the most significant challenges is the high volume of water lost in distribution systems—developing countries, for instance, lose an estimated 32 billion cubic meters of treated water annually from urban supply networks [46].

Smart Water Distribution Networks (SWDNs) address these issues by leveraging IoT technologies to enable real-time monitoring, data collection, and control within WDNs. In-

tegrated with LPWANs, IoT devices offer broad coverage and energy efficiency, making them well-suited for widespread deployment in water networks. These technologies provide continuous monitoring of critical parameters such as flow rates, pressure, and water quality, ensuring the reliability and safety of the water supply [11].

WSNs, an essential component of this infrastructure, are widely employed in various sectors, from environmental monitoring to agriculture, where they regulate water usage and track service delivery [47]. Their role in collecting and transmitting data is vital for managing water resources efficiently and sustainably.

The primary aim of smart water management is to gather actionable insights into a city's water supply, pressure, and distribution, optimizing both water facilities and the electricity used in water movement [45]. As challenges like economic growth, climate change, and population increase strain water resources, ICT plays a key role in improving water management, reducing waste, and ensuring water quality through advanced technology.

The integration of IoT into WDSs automates the detection of malfunctions and leakages [48], converting environmental data into actionable insights. Time-sensitive data are processed in real-time, while less urgent data are stored in the cloud to prevent network congestion. At the network level, this data is converted into digital streams for further analysis [49], while the user-facing layer provides end-users with real-time control and insights into the water distribution system. These optimizations align with the goals of the Green IoT framework, which enhances traditional methods by introducing environmentally friendly, energy-efficient solutions that improve sustainability and operational efficiency.

1.3.3 Challenges in Smart Water Distribution Networks

While SWDNs represent a significant leap forward in water management, several inherent challenges must be addressed to fully harness their potential. These challenges span multiple areas, each critical to the effective operation and sustainability of the network.

Sampling Design

Accurate data collection is the cornerstone of effective WDN management, and this begins with the design of the sampling framework. The strategic placement of sensors is essential for precise hydraulic and water quality modeling. This complex process often relies on sensitivity and covariance matrices to determine optimal locations, with initial estimates of pipe roughness playing a pivotal role in guiding the design [11].

Leak Assessment and Control

Leakage within WDSs poses a significant threat to resource sustainability and service quality. The detection and control of leaks are therefore critical components of SWDN management. Traditional leak detection methods frequently struggle to identify small or slow leaks, underscoring the need for advanced techniques. Integrating AI-driven data analysis with real-time

monitoring systems enhances the accuracy and responsiveness of leak detection efforts, ultimately reducing water loss and improving system efficiency [12].

Water Quality and Security

Ensuring the quality and security of water within WDNs is a fundamental challenge. Conventional approaches typically involve the periodic sampling and analysis of microbial and physicochemical parameters. However, these methods may not be sufficient to detect rapid or unexpected changes in water quality. The incorporation of online water quality sensors, complemented by AI-based anomaly detection systems, provides a more robust solution. These technologies not only improve the ability to monitor and respond to potential contamination events in real-time but also bolster the overall security of the water supply against physical, chemical, biological, and cyber threats [11].

Pressure Management

Managing water pressure within a WDN is crucial for minimizing leakage and ensuring that consumers receive water at adequate pressures. Pressure-reducing valves (PRVs) are commonly used to maintain pressure levels, but their optimal placement and configuration present a significant challenge. The costs associated with installing and maintaining PRVs, coupled with the complexity of balancing leakage reduction with sufficient pressure maintenance, require sophisticated optimization strategies. AI-based techniques offer promising solutions, enabling the precise configuration of PRVs to achieve the desired balance and enhance the network's overall performance [50].

1.4 EPANET and WNTR

EPANET [51] is an open-source software tool used to simulate the hydraulic and water quality behavior of pressurized WDNs over extended periods. These networks consist of pipes, junctions, pumps, valves, and storage facilities such as tanks or reservoirs. EPANET tracks key parameters, including water flow in pipes, pressure at nodes, water levels in tanks, and the concentration of chemical species throughout the network across multiple time steps. The software also supports simulations of water age and source tracing.

Primarily used for research, EPANET aids in understanding the movement and fate of water constituents within distribution systems. It supports various applications, including sampling program design, hydraulic model calibration, chlorine residual analysis, and consumer exposure assessment. The software is capable of evaluating different management strategies for water quality improvement, such as:

- Modifying source utilization in multi-source systems,
- Adjusting pumping and tank operation schedules,

- Implementing satellite treatments like re-chlorination,
- Targeted cleaning and replacement of pipes.

EPANET provides an integrated environment for data editing, simulation execution, and result visualization in formats like color-coded maps, data tables, time series graphs, and contour plots.

1.4.1 Hydraulic Modeling Capabilities

EPANET includes a hydraulic analysis engine capable of handling networks of any size. It calculates friction head loss using the Hazen-Williams, Darcy-Weisbach, or Chezy-Manning formulas [52] and accounts for minor losses due to bends and fittings. The software models pumps with either constant or variable speeds, computes pumping energy and costs, and supports various valve types, including shutoff, check, pressure-regulating, and flow control valves. Storage tanks can be modeled with variable diameters, and multiple demand categories with time-variable patterns can be defined at nodes. The system operation can be controlled by simple tank level or timer controls, as well as by complex rule-based controls.

1.4.2 Water Quality Modeling Capabilities

EPANET's water quality modeling features include tracking the movement of non-reactive and reactive substances, such as disinfection by-products and chlorine residuals. It can simulate water age, trace the flow percentage from specific nodes, and model reactions in both the bulk flow and at pipe walls. The software uses n-th order kinetics for bulk flow reactions and zero or first-order kinetics for pipe wall reactions, accounting for mass transfer limitations. Reaction rates can be modified on a pipe-by-pipe basis, and wall reaction rates can be correlated with pipe roughness. Additionally, EPANET allows time-varying inputs of concentration or mass at any network location and models storage tanks as complete mix, plug flow, or two-compartment reactors.

These features enable the study of phenomena such as source blending, water age distribution, chlorine residual decay, disinfection by-product formation, and contaminant propagation.

1.5 WNTR

Water Network Tool for Resilience (WNTR) [53, 54] extends the capabilities of EPANET by addressing a broader range of hydraulic and water quality scenarios, particularly those relevant to disruptive events and recovery processes. While EPANET models pressures and flows in WDNs, WNTR adds the ability to simulate both demand-driven and pressure-dependent demand conditions: as show in Figure 1.3, Pressure-Driven Analysis (PDA) adjusts water demand based on the available pressure, reducing demand when pressure is insufficient, while

Demand-Driven Analysis (DDA) assumes the full demand is always met, regardless of the available pressure.

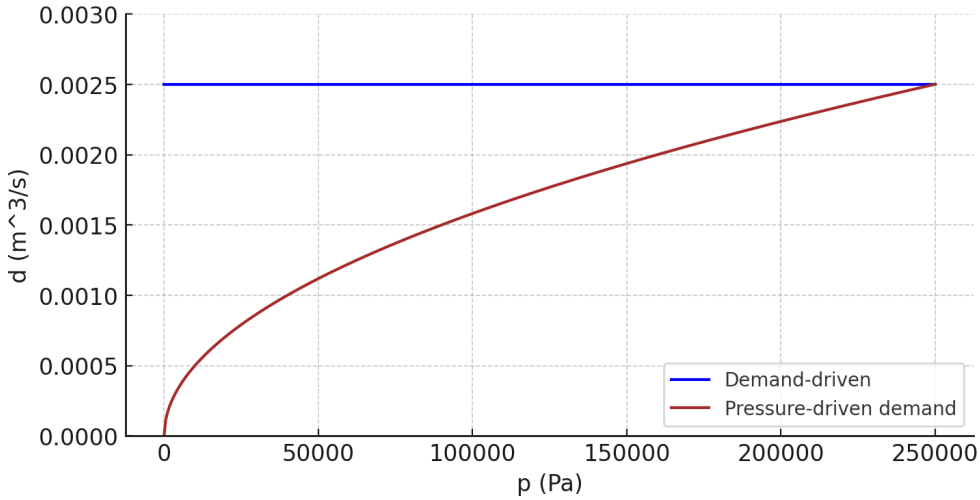


Figure 1.3: A simulated example of pressure vs. demand using demand-driven and pressure-driven models.

WNTR also generates damage states by sampling from fragility or survival curves. It allows users to assign damage probabilities to individual components such as pipes, pumps, and tanks, based on a variety of statistical distributions. This enables the simulation of stochastic disaster scenarios, where users can also assign probabilities to response and repair strategies, such as repair times and the availability of crews.

Within WNTR, simulation results support resilience analysis through various metrics, including hydraulic, topographic, water quality, and cost metrics. Examples include water service availability, contamination levels, betweenness centrality, and the identification of articulation points. WNTR includes graphing capabilities that visualize results through network graphics, time-series plots, animations, and interactive graphics.

WNTR can create water network models directly from EPANET-formatted input files (EPANET INP). Users can modify the network structure and operations within WNTR and save the updated model back into an EPANET INP file, allowing easy transition between EPANET and WNTR.

WNTR is built in Python and integrates several widely-used open-source Python packages, such as Numpy and Scipy for numerical computations, Pandas for time-series analysis, NetworkX for topographic analysis, and Matplotlib and Plotly for graphics and animations. These tools provide flexibility in building custom analyses, from simple network model generation to complex stochastic resilience assessments based on specific disruption and recovery scenarios.

For quality assurance, WNTR uses Travis CI for continuous integration, running automated tests whenever the code repository is updated. Additionally, WNTR is tested on private servers using large water utility network models before each software release.

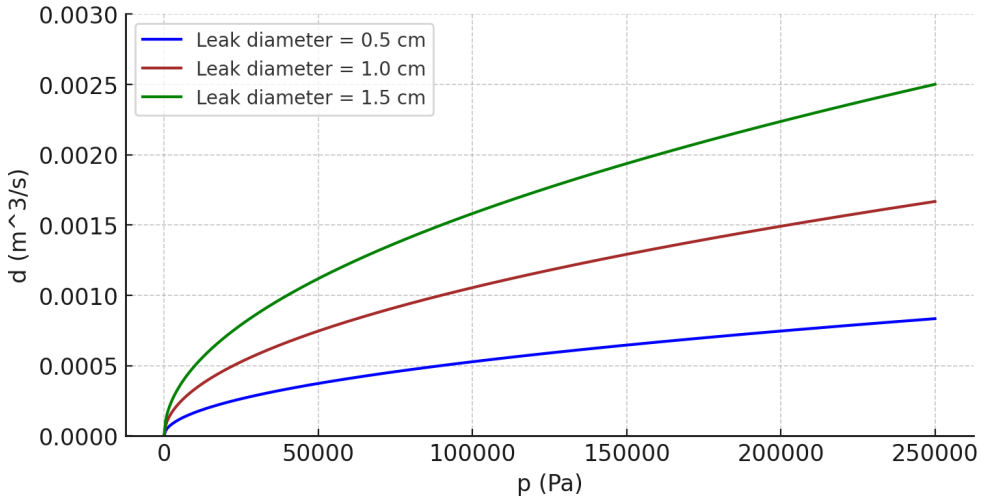


Figure 1.4: Example of the relationship between leak demand and pressure.

1.5.1 Leak Model

The WNTRSimulator includes a feature to model leaks within a network using a specific leak model, differing from the EpanetSimulator where emitter coefficients are employed. In WNTRSimulator, emitter coefficients defined in the water network model options are not utilized. However, users can still model leaks using the EpanetSimulator by defining these coefficients.

In WNTRSimulator, leaks are modeled using a general form of the equation proposed by Crowl and Louvar, where the mass flow rate through a leak is represented as:

$$d_{\text{leak}} = C_d A p^{\alpha} \sqrt{\frac{2}{\rho}}$$

where:

- d_{leak} is the leak demand (m³/s),
- C_d is the discharge coefficient (dimensionless),
- A is the area of the hole (m²),
- α is an exponent related to the leak characteristics (dimensionless),
- p is the gauge pressure (Pa),
- ρ is the fluid density (kg/m³).

For cases where $\alpha = 0.5$ (assuming large leaks from steel pipes), the equation simplifies to:

$$d_{\text{leak}} = C_d A \sqrt{2gh}$$

where:

- g is the acceleration due to gravity (m/s²),
- h is the gauge head (m).

The default discharge coefficient C_d is 0.75, assuming turbulent flow, but this can be adjusted by the user. The exponent α is set to 0.5 by default and cannot currently be changed.

Leaks can be introduced at junctions and tanks. A pipe break is modeled by specifying a leak area large enough to effectively drain the pipe. WNTR also provides functionality to add leaks at any location along a pipe by splitting the pipe into two sections and inserting a node at the leak site. As shown in Figure 1.4, WNTR models leak demand as pressure-dependent outflow at network nodes or pipes. The relationship between pressure and leak demand is typically non-linear, with the leak rate increasing as pressure rises. This enables the simulation of pressure-dependent leaks, reflecting real-world conditions where higher pressure results in greater water loss.

1.6 Machine Learning

ML, a branch of AI, has experienced rapid growth, particularly in data analysis and intelligent computing [55, 56]. ML enables systems to learn from experience without explicit programming, making it fundamental to modern intelligent systems. It is also a key driver of the fifth industrial revolution (Industry 5.0) [57], where automation and smart technologies enhance traditional industrial practices [58, 59, 60].

1.6.1 ML and Data

MLs algorithms are designed to process data and uncover patterns related to individuals, business processes, and events. The availability of relevant data is crucial for building effective MLs models [58, 59]. Various datasets are widely used across different application domains, such as cybersecurity (e.g., NSL-KDD [61], UNSW-NB15 [62]), smartphone usage (e.g., call logs [63, 64], SMS logs [65]), IoT data [66, 67, 68], and health data (e.g., heart disease [69], diabetes [70, 71], and COVID-19 [72, 73]).

Although the type of data varies by domain, the goal is consistent: to analyze data within a specific context, extract valuable insights, and build intelligent systems. The choice of MLs techniques depends on the nature of the data and the task at hand.

1.6.2 Learning Techniques

MLs training algorithms are typically classified into four main categories: supervised, unsupervised, semi-supervised, and reinforcement learning [74]. These methods address various problem types and data structures, making them suitable for a wide range of real-world applications.

- **Supervised learning** trains models on datasets containing both input features and corresponding labeled outputs. The model learns to map inputs to outputs based on this labeled data [75]. Supervised tasks are typically divided into two main categories:
  - *Classification tasks*, where the model assigns categorical labels to data points (e.g., classifying users as ”buyers” or ”non-buyers” based on their behavior).
  - *Regression tasks*, which predict continuous values (e.g., forecasting sales figures) [59].
- **Unsupervised learning**, by contrast, operates on unlabeled data, aiming to identify patterns, structures, or groupings without human intervention [75]. Common tasks include:
  - *Clustering*, which groups data points based on similarities.
  - *Dimensionality reduction*, which simplifies high-dimensional datasets while retaining key information [76].
- **Semi-supervised learning** combines labeled and unlabeled data, useful in tasks like fraud detection and machine translation [74, 59].
- **Reinforcement learning** optimizes decision-making through feedback, with applications in robotics, autonomous vehicles, and system optimization [77, 74, 78].

Selecting the appropriate ML technique depends on the specific problem and the nature of the available data. Each approach offers unique benefits for building models across a variety of domains.

1.6.3 In-depth Overview of Supervised Machine Learning Tasks

MLs supervised tasks are primarily categorized based on their goals, and classification is one of the fundamental tasks within supervised learning.

Classification

In classification, the goal is to predict a class label for a given input, mapping input variables (X) to an output variable (Y) [75]. We can have different types of classifications, mostly depending on the number of classes to predict.

- **Binary classification**: Involves two class labels, typically representing a normal and abnormal state, such as ”cancer detected” vs. ”cancer not detected,” or ”spam” vs. ”not spam” [75].
- **Multiclass classification**: Assigns data points to one of several categories. For example, in the NSL-KDD dataset [61], network attacks are classified into categories like DoS, U2R, R2L, and Probing.

- **Multi-label classification**: Allows data points to belong to multiple categories simultaneously, useful in cases like categorizing news articles by city, topic, and type (e.g., ”technology” and ”latest news”) [79].

Several algorithms are widely used to address classification problems in ML:

- **Naive Bayes (NB)**: Based on Bayes' theorem, Naive Bayes assumes feature independence, performing well in tasks like text classification and spam filtering with minimal data. Common variants include Gaussian, Multinomial, and Bernoulli [80, 79], though its performance may suffer in complex datasets due to its strong independence assumptions.
- **Support Vector Machine (SVM)**: SVM separates classes by constructing a hyperplane in high-dimensional space, with kernel functions (e.g., linear, polynomial, RBF) for linear and non-linear problems [81]. It is effective but can struggle with noisy or overlapping data [79].
- **Decision Tree (DT)**: DTs classify data by splitting it based on feature values, with well-known algorithms like ID3, C4.5, and CART [82, 83, 84]. They are easy to interpret but prone to overfitting without pruning [79].
- **Stochastic Gradient Descent (SGD)**: An iterative optimization method for high-dimensional spaces, SGD reduces computational cost by randomly selecting data subsets. Effective for large-scale tasks like text classification, it requires careful tuning of hyperparameters like regularization and iterations [75, 79].

Each algorithm has its strengths and weaknesses, and the choice depends on data characteristics and the problem at hand [85].

Regression

Regression analysis predicts continuous values (y) based on one or more predictor variables (x) [75], unlike classification, which predicts discrete labels. It is commonly applied in areas such as financial forecasting, cost estimation, and time series analysis.

- **Simple and Multiple Linear Regression**: Linear regression models the relationship between a dependent variable and one or more independent variables.
  - In **Simple Linear Regression**, the relationship is expressed as:

$$y = a + bx + e$$

    where a is the intercept, b is the slope, and e is the error term.
  - In **Multiple Linear Regression**, this is extended to multiple predictor variables:

$$y = a + b_1x_1 + b_2x_2 + \cdots + b_nx_n + e$$

This allows for more complex predictions by incorporating multiple factors.

1.6.4 Real-World Applications of Machine Learning

In the era of the Fourth Industrial Revolution (4IR), ML has gained widespread adoption across various domains.

Predictive analytics and decision-making: ML plays a key role in predictive analytics, enabling intelligent decision-making by identifying relationships between variables to forecast outcomes [86, 87, 75]. In e-commerce, for example, ML helps retailers understand consumer preferences, optimize inventory, and improve logistics and warehousing. Algorithms such as Decision Trees (DTs), SVMs, and neural networks (NNs) are frequently used for these tasks [85, 88].

Cybersecurity and threat intelligence: As cyber threats become increasingly complex, ML is essential in identifying patterns in data to detect and respond to malware, especially in encrypted traffic [60]. Its continuous learning capabilities make it a key technology in protecting networks, systems, and data from digital attacks.

Internet of Things (IoT) and smart cities: The IoT connects everyday objects, allowing them to share data and automate tasks without human intervention [60]. In smart cities, ML helps optimize services like traffic prediction, energy management, and public safety, improving residents' overall quality of life [58]. From predicting parking availability to energy consumption, ML enables context-aware, timely decision-making for city management [87].

Traffic prediction and transportation: ML is vital in predicting traffic patterns, essential for managing congestion, reducing fuel consumption, and lowering emissions in modern cities [89]. deep learning (DL) models can provide accurate traffic forecasts, which are key components of intelligent transportation systems [90, 91, 92].

Image, speech, and pattern recognition: ML excels in recognizing images, speech, and patterns, with applications ranging from medical diagnosis (e.g., detecting cancer in x-rays) to social media tagging and face recognition [93].

Sustainable agriculture: ML supports sustainable agriculture by helping farmers make data-driven decisions that increase productivity while minimizing environmental impact [94]. IoT technologies and mobile devices collect vast amounts of data, which ML models analyze to optimize agricultural practices [95, 96, 97].

1.6.5 IoT and Machine Learning

The convergence of machine learning and IoT marks a significant technological advancement, particularly for resource-limited devices, where efficiency, accuracy, and cost savings are paramount [37]. Machine learning enhances both the communication and computational capabilities of IoT systems, enabling more efficient control and data-driven decision-making. With the deployment of billions of sensors and enhanced communication infrastructures, IoT

offers transformative opportunities for both improving human life quality and fostering industrial innovation, particularly in the realm of Industry 5.0 [57].

Machine learning plays a crucial role by enabling IoT systems to process large volumes of sensory data and derive actionable insights. This integration allows for faster and more precise operational decisions, which are essential for addressing the growing complexity of IoT-related communication and computational tasks. The increasing relevance of IoT data analytics is a direct outcome of these advancements, highlighting machine learning's pivotal role in unlocking IoT's full potential.

As IoT adoption continues to expand, the volume of data generated by distributed devices is expected to increase dramatically [37]. This surge is driven by the integration of IoT in critical applications, necessitating efficient and intelligent data analytics to extract meaningful insights and predict future system states.

1.7 Neural Networks and Deep Learning

NNs, inspired by biological neural systems, are a core class of machine learning models. They consist of interconnected layers of artificial neurons, with the architecture—specifically the number of layers and neurons—being crucial to performance [98, 76]. DL, a subset of NNs, utilizes multiple hidden layers to automatically learn complex patterns from high-dimensional data such as images, text, and audio [99].

A key strength of NNs is their ability to discover intricate, nonlinear relationships without manual hypothesis design, making them particularly effective for large datasets, where they often outperform shallower models [59, 100]. However, for simpler tasks or those requiring interpretability, traditional models may still be preferable [101, 102]. Despite these capabilities, NNs have limitations in understanding context and intent [103].

Building on foundational concepts like linear regression—where predictions follow $W^T X + b$—NNs employ networks of interconnected neurons arranged in layers. These layers, often consisting of hundreds or thousands of neurons, process data at various levels of abstraction, enabling deep learning models to handle complex patterns and large-scale datasets.

1.7.1 Activation Functions

Activation functions are a vital component of NNs, introducing nonlinearity that allows the network to model complex patterns. Inspired by biological neurons, which either fire or remain inactive, these functions help transform input data into higher-dimensional spaces, enhancing separability and enabling effective classification. The sigmoid function, a commonly used activation function, is a key example. In tasks like logistic regression, the sigmoid function transforms input values into outputs between 0 and 1, making it well-suited for binary classification problems.

1.7.2 Parameter Learning

NN models, and more specifically deep neural networks (DNNs) (i.e., deeper networks with more than one layer), like traditional ML models, rely on optimization techniques such as gradient descent to learn parameters. Gradient descent is particularly effective for convex functions, where a single global minimum or maximum exists. However, the optimization of DNNs is often non-convex, meaning multiple local optima exist, complicating the learning process. The network adjusts its parameters by minimizing the error between predicted and actual values, iteratively refining its predictions as learning progresses.

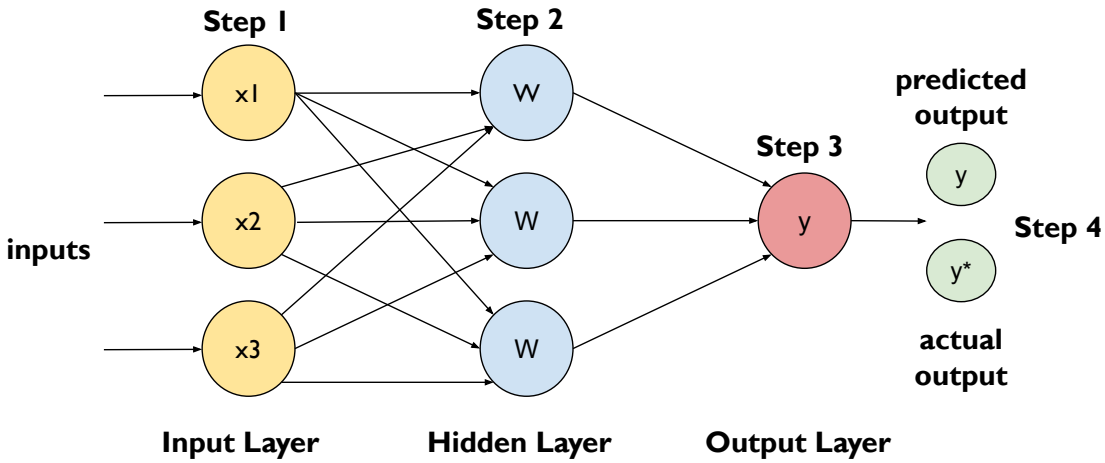


Figure 1.5: A simple neural network: (step 1) inputs (x_1, x_2, x_3) are weighted and passed through the hidden layer (step 2), producing a prediction ($\hat{y}$) at the output layer (step 3). The error, calculated as the difference between the predicted output ($\hat{y}$) and the actual output (y), is then propagated back through the network to adjust the weights (step 4).

1.7.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of DNN models designed to handle complex tasks such as image recognition and voice data analysis [104, 105]. These networks consist of multiple layers of two-dimensional planes of neurons, where each neuron is sparsely connected to a local region of the previous layer, which enhances computational efficiency compared to traditional NN [104].

Convolution:

In a convolution layer, the activation value a_j^l of the j^{th} feature map is given by:

$$a_j^l = f\left(b_j^l + \sum_{i \in M_j^l} a_i^{l-1} * k_{ij}^l\right),$$

where f is a non-linear activation function, $*$ denotes the two-dimensional convolution operator, b_j^l is the bias term for the j^{th} feature map, and k_{ij}^l is the convolution kernel applied to the

i^{th} feature map from the previous layer. Weight sharing is employed to reduce the number of parameters, as the same weights are used across all locations in a feature map [104].

Subsampling:

Each convolutional layer is followed by a subsampling (or pooling) layer, which reduces the spatial dimensions of the feature maps. This operation, defined as:

$$a_j^l = \text{down}(a_i^{l-1}, N_l),$$

where N_l is the subsampling boundary, allows the network to retain relevant features while achieving invariance to small shifts or distortions in the input data. Subsampling thus enhances robustness without changing the essential characteristics of the output [104].

Output Layer:

The final output of the network, after passing through fully connected layers, is given by:

$$\text{output} = f(b^0, W^0 f_v),$$

where b^0 is the bias vector, W^0 is the weight matrix, and f_v is the eigenvector. The model parameters include b_j^l, k_{ij}^l, and W^0, with the alternating convolution and subsampling layers progressively reducing spatial resolution while increasing the number of feature maps [104].

Training Process:

The training of CNNs involves two main phases: forward propagation and backpropagation (Figure 1.5). These processes work together to iteratively refine the model's parameters, optimizing its performance over time.

- **Forward Propagation:** In this phase, input data is passed through the network layer by layer. At each stage, the data is transformed via convolutional, activation, and subsampling layers, ultimately producing an output prediction. The process includes:

  1. Randomly selecting a batch of samples from the training dataset.
  2. Passing the selected samples through the network to compute activations for each layer.
  3. Generating the network's output based on these activations.

- **Backpropagation:** Once the output is generated, the network computes the error between the predicted output and the true labels using a predefined loss function (e.g., cross-entropy or mean-squared error (MSE)). The error is then propagated backwards through the network, adjusting the weights in each layer. Specifically, the process involves:

1. Calculating the loss by comparing the predicted and actual outputs.
2. Backpropagating the gradients through each layer, using the chain rule to compute the gradient of the loss with respect to the network's weights.
3. Updating the network's weights based on the computed gradients, typically through optimization methods such as stochastic gradient descent (SGD).

This cycle of forward and backward propagation is repeated iteratively, with the objective of minimizing the loss function over successive training iterations. The process continues until convergence, where changes in the loss function become negligible [104].

A notable early example of CNNs is the LeNet-5 architecture, developed for digit recognition on the MNIST dataset, where it achieved a 0.9% error rate [106]. More recently, the introduction of graphic processing unit (GPU) acceleration has significantly reduced CNN training times, with Krizhevsky et al. [106] using CNNs to achieve breakthrough results on the ImageNet dataset. These advancements have solidified CNNs as the state-of-the-art model for high-dimensional data, enabling direct input of raw images and eliminating the need for complex feature engineering.

CNNs are highly effective for tasks involving image and speech recognition due to their weight-sharing architecture. Moreover, their ability to handle scale, rotation, and translation invariance makes them particularly robust against a wide range of input variations.

1.7.4 Graph Neural Networks

The rise of DL across various fields is largely due to advancements in computational power (e.g., GPUs), the availability of large-scale datasets, and its capability to uncover latent patterns from structured Euclidean data, such as images, text, and videos. For example, in image analysis, images are represented as regular grids in Euclidean space. CNNs are particularly effective in extracting meaningful local features by leveraging shift-invariance, local connectivity, and compositionality [107].

However, many emerging applications involve data represented as graphs, where the relationships between entities are crucial. In e-commerce, for instance, graph-based models leverage user-product interactions to provide accurate recommendations, while in chemistry, molecular graphs are used to predict bioactivity for drug discovery. These applications highlight the limitations of traditional DL models, which are highly effective for Euclidean data but struggle with the irregular nature of graphs.

Graph data presents unique challenges due to its irregularity: graphs consist of unordered nodes with variable connectivity patterns, making operations like convolutions—straightforward in the image domain—more difficult to apply. Additionally, traditional machine learning algorithms often assume data point independence, an assumption that does not hold for graphs where nodes are interconnected by various relationships, such as citations, social connections, or interactions.

To overcome these challenges, new techniques have emerged to handle graph data effectively [108]. The early development of Graph Neural Networks (GNNs) began with NNs applied to directed acyclic graphs [109]. This evolved into Recurrent Graph Neural Networks (RecGNNs), where node representations are iteratively refined by propagating information from neighboring nodes until a stable state is reached [110], [111], [112]. Although this approach can be computationally expensive, recent efforts have focused on optimizing these processes [113], [114].

GNN Outputs

The outputs of GNNs can be applied to a variety of graph analytics tasks, which are typically categorized into three primary levels:

- **Node Level**: At this level, GNNs are used for tasks such as node regression and classification. By propagating information across the graph through convolutions, GNNs extract meaningful node representations. A final layer, such as a softmax or multilayer Multilayer Perceptron (MLP), is applied to enable end-to-end learning for node-level predictions.
- **Edge Level**: For edge classification and link prediction tasks, GNNs use the hidden representations of two connected nodes. A similarity function or a NN can then be applied to predict the label or strength of the connection between them.
- **Graph Level**: At the graph level, GNNs handle tasks such as graph classification. They employ pooling and readout operations to generate a compact graph representation. Pooling reduces the graph to its key substructures, while the readout layers aggregate node-level features into a single output representing the entire graph.

GNN Training Frameworks

GNNs can be trained using supervised, semisupervised, or unsupervised methods, depending on the task and the availability of labeled data. These approaches cater to different learning scenarios within graph analytics.

- **Semisupervised Learning (Node-Level Tasks)**: In the case of partially labeled graphs, semisupervised learning allows Graph Convolutional Neural Networks (GCNNs) to leverage both labeled and unlabeled nodes for tasks like classification and regression. By stacking graph convolutional layers, GNNs propagate information across the graph, enabling predictions for unlabeled nodes. This is typically followed by a softmax for classification or a linear layer for regression [115].
- **Supervised Learning (Graph-Level Tasks)**: When sufficient labels are available, supervised learning is applied to entire graphs for tasks such as graph-level classification

or regression. GCNNs achieve this by combining graph convolutional layers with pooling and readout operations. Pooling reduces the graph size by focusing on important substructures, and readout layers aggregate node features into a compact representation. The final prediction is then made using a multilayer MLP, followed by a softmax or regression layer [116, 117, 118, 119].

- **Unsupervised Learning (Graph Embedding)**: In unsupervised settings, where no labeled data is available, GNNs are employed to learn graph embeddings. One popular approach uses an autoencoder framework, where graph convolutional layers act as the encoder, and a decoder reconstructs the graph structure [120, 121]. Another common method is negative sampling, which uses logistic regression to differentiate between positive (existing links) and negative (randomly generated) node pairs [122].

Inspired by the success of CNNs in computer vision, many GCNNs methods have been proposed to process graph data efficiently. These approaches are broadly divided into spectral-based and spatial-based methods, each offering different strategies for leveraging graph structure in learning tasks.

Spectral-based Approaches

Spectral-based GCNNs are founded on graph signal processing principles, where undirected graphs are represented by the normalized graph Laplacian $L = I_n - D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$, with D as the diagonal node degree matrix. The eigenvectors of L form an orthonormal basis that enables the graph Fourier transform $F(x) = U^T x$ and its inverse $F^{-1}(\hat{x}) = U\hat{x}$, which project graph signals into and out of the spectral domain. In this domain, graph convolution is defined as $x * g = U g_\theta U^T x$, where g_θ is a learnable filter [123].

While Spectral CNNs are effective, they come with several limitations. Perturbations to the graph can alter the eigenbasis, making filters highly specific to the graph's structure, and the computational cost of eigendecomposition scales with $O(n^3)$. To mitigate these issues, ChebNet [124] approximates the filters using Chebyshev polynomials, significantly reducing the complexity to $O(m)$. The convolution approximation is expressed as $x * g_\theta = \sum_{i=0}^{K} \theta_i T_i(\tilde{L})x$, where T_i are Chebyshev polynomials computed recursively, allowing efficient localized feature extraction independent of the graph's size.

Spatial-based Approaches

Spatial-based GCNNs perform graph convolutions by aggregating information from a node's neighbors, similar to how CNNs aggregate information from adjacent pixels in an image. In this approach, each node updates its representation by combining its own features with those of its neighbors, effectively capturing the local graph structure through message passing.

The general operation for spatial-based convolution is given by:

$$h_v^{(k)} = \sigma\left(W^{(k)} h_v^{(k-1)} + \sum_{u \in \mathcal{N}(v)} W^{(k)} h_u^{(k-1)}\right),$$

where $h_v^{(k)}$ represents the node v's features at layer k, $\mathcal{N}(v)$ is the set of neighbors of node v, and $W^{(k)}$ is a learnable weight matrix. This operation mimics message passing, where nodes propagate information through their connections.

Extensions of this basic approach, such as NN4G [125] and GraphSAGE [122], enhance the process by either summing neighbor features directly or using sampling techniques to handle large-scale graphs. A more comprehensive formulation is provided by Message Passing Neural Networks (MPNNs) [126], which formalize the message-passing process as:

$$h_v^{(k)} = \text{Update}\left(h_v^{(k-1)}, \sum_{u \in \mathcal{N}(v)} \text{Message}(h_v^{(k-1)}, h_u^{(k-1)}, e_{vu})\right),$$

where node updates depend on the messages exchanged between nodes, potentially incorporating edge features e_{vu}.

Spatial-based GCNNs are widely used in tasks such as node classification and graph classification, where localized feature aggregation is crucial [126, 122].

Comparison Between Spectral and Spatial Models

Spectral models are grounded in graph signal processing and typically involve designing graph filters (e.g., Cayleynets [127]) to build GCNNs. Despite their solid theoretical foundation, spatial models are generally preferred for their superior efficiency, scalability, and flexibility [108].

- **Efficiency**: Spectral models require expensive eigenvector computations or necessitate processing the entire graph at once, which becomes inefficient for large-scale graphs. In contrast, spatial models perform localized convolutions through message passing, allowing computations to be done in parallel for batches of nodes. This makes spatial models significantly more scalable.

- **Generality**: The reliance on a fixed graph structure limits spectral models' ability to generalize to new graphs, as changes to the graph alter the eigenbasis. Spatial models, by working on local node neighborhoods, can easily share weights across different graphs, enabling them to generalize more effectively to varying structures.

- **Flexibility**: Spectral models are typically restricted to undirected graphs due to their reliance on graph Laplacians. Spatial models, however, are more flexible and can handle a wide range of graph types, including directed graphs [128, 129], signed graphs [130], and heterogeneous graphs [131, 132]. This flexibility arises from the ability of

spatial models to incorporate these structures into the message-passing aggregation function.

GINEConv

The Graph Isomorphism Network with Edge features (GINEConv) [133, 134] is a variant of graph convolutional layers designed to incorporate edge information into the message passing framework of GNNs. GINEConv extends the standard Graph Isomorphism Network (GIN) by integrating edge attributes into the node update process, making it more effective for tasks where edge features are crucial.

The node update rule in GINEConv is defined as:

$$\mathbf{h}_i^{(k+1)} = \text{MLP}^{(k)} \left((1+\epsilon) \cdot \mathbf{h}_i^{(k)} + \sum_{j \in \mathcal{N}(i)} \text{ReLU}\left(\mathbf{h}_j^{(k)} + \mathbf{e}_{i,j} \right) \right),$$

where $\mathbf{h}_i^{(k)}$ represents the feature vector of node i at layer k, $\mathcal{N}(i)$ denotes the set of neighbors of node i, and $\mathbf{e}_{i,j}$ is the edge feature between node i and its neighbor j. The parameter ϵ is learnable and allows the model to adjust the contribution of the central node's previous features.

The (MLP) is responsible for ensuring the model's expressiveness in capturing complex graph structures and interactions. By incorporating edge features $\mathbf{e}_{i,j}$, GINEConv is particularly well-suited for applications such as molecular graphs, where bond information between atoms is crucial.

This formulation strengthens the ability of GNNs to differentiate between non-isomorphic graphs, offering greater discriminative power compared to standard GNNs variants.

1.8 Adversarial Machine Learning

ML algorithms are generally designed to generalize from a training dataset to an unseen test dataset. While there are many methods for data collection and annotation, a common assumption is that both datasets are generated by a stationary stochastic process. However, this assumption often oversimplifies real-world scenarios. Early ML research recognized this issue, particularly when evaluating and enhancing classifier robustness. Robustness in this context refers to the ability of a classifier to withstand anomalies in the training or test data [135]. These anomalies may not always be rare events; in many cases, they result from deliberate attacks by adversarial parties seeking to manipulate the system.

In high-stakes domains like malware classification [136], adversarial examples pose significant challenges to ML-based systems [136, 137]. For instance, an attacker may slightly alter a malware file, causing it to be misclassified as benign while retaining its malicious functionality. Such vulnerabilities are particularly concerning in safety-critical applications, including unmanned aircraft collision avoidance systems [138] and self-driving cars [139],

where even minor classification errors could have severe consequences.

Adversarial machine learning (AML) involves an interaction between two adversarial parties: the attacker, who generates adversarial examples, and the defender, typically a classifier trained using ML algorithms. The goal of the attacker is to exploit the classifier's weaknesses by crafting examples that deceive the model. Meanwhile, the defender's objective is to build a system robust enough to withstand even the most challenging attack scenarios. As Huang et al. [140] note, AML focuses on developing strategies to defend against these sophisticated attacks. Although the attacker and defender's goals may seem opposed, both rely heavily on understanding the other's strategies. Attackers use knowledge of the defense mechanisms to guide their attacks, while defenders must design systems that are resilient under the worst-case conditions.

1.8.1 Background on Adversarial Machine Learning

Recent research in the DL field has refined the definition of "adversarial," focusing specifically on the creation of small, imperceptible perturbations to normal examples, which can deceive ML algorithms. This refers to subtle modifications that may be invisible to humans but can significantly alter the output of a classifier. Formally, a classification function $F : \mathbb{R}^m \to \{1, \ldots, k\}$ maps real-valued input vectors to discrete label sets. In DNNs, the classifier's output is typically determined by a probability distribution produced by a function f, with the final class prediction made via a softmax layer. For a given input x, the NN's prediction is expressed as $F(x) = \arg\max_j f_j(x)$. The adversary's goal is to find a slightly modified input x' that is close to x, but for which $F(x') \neq F(x)$.

The network's parameters, θ, are optimized by minimizing the expected value of a loss function, $\mathbb{E}_{x\sim\text{data}}[\text{loss}_{f,l}(x)]$, where f outputs a probability vector q, with each component $q_i = \text{softmax}(i)$, and l is the expected class label for the input x. The expected label is typically represented as a Dirac distribution p, defined as:

$$p_i = \begin{cases} 1 & \text{if } i = l \\ 0 & \text{otherwise} \end{cases}$$

For classification tasks, the loss function is often based on cross-entropy, $H(p, q) = \mathbb{E}_p[-\log q]$, which is equivalent to the Kullback-Leibler divergence between p and q, assuming p is fixed.

The origins of adversarial research in DNNs trace back to the pivotal work by Szegedy et al. [141]. Their study revealed that even tiny, imperceptible perturbations in input images could lead to dramatic and seemingly inexplicable misclassifications in DNN-based computer vision systems.

White-box and Black-box Attacks

The primary distinction between white-box and black-box attacks lies in the attacker's knowledge of the classifier's internal workings. In a white-box attack, the attacker has full access

to the classifier's architecture and parameters, allowing for precise adversarial perturbations that remain undetectable to human observers. This contrasts with black-box attacks, where the attacker lacks direct access to the classifier's internals and must infer its behavior indirectly. Barreno et al. [142] partially capture this difference through the concept of "influence dimensions." However, while Barreno's causative attacks aim to manipulate the classifier during its training phase, white-box attacks typically operate during inference and often rely on the gradient of the cost function with respect to the input x.

Black-box attacks, on the other hand, require attackers to probe the classifier by analyzing its responses to various inputs. This form of attack is akin to the exploratory attacks described by Barreno et al. The specific assumptions about the information returned by the classifier can vary. For instance, in the setting described by Papernot et al. [143], the classifier only provides the predicted class label, whereas in the work of Narodytska and Kasiviswanathan [144], partial access to the classifier's confidence scores is allowed. In this paper, we consider the distinction between white-box and black-box attacks to be the most significant factor when discussing attack strategies.

Targeted and Untargeted Attacks

In contemporary terms, a targeted attack refers to an attack where the adversary forces the classifier to misclassify an input into a specific target class t. In traditional network security contexts, such as intrusion detection, where binary classification is prevalent, the target output is often implied. However, in tasks like image classification, which may involve thousands of possible classes, targeting a specific output class becomes more relevant and challenging. A variation of targeted attacks focuses on removing the correct label from the top-k ranked outputs of the classifier, as demonstrated by Narodytska and Kasiviswanathan [144].

Conversely, untargeted attacks aim to cause misclassification without dictating a specific incorrect class. A notable example is the work by Moosavi-Dezfooli et al. [145], who showed that a single, small perturbation could fool a DNN across a wide range of natural images, effectively leading to misclassification regardless of the correct class.

Least-Cost and Constrained Attacks

Adversarial attacks seek to manipulate a model's predictions by introducing minimal perturbations to the input data. These attacks involve balancing the magnitude of the perturbation and the effectiveness of the attack, typically measured by changes in the classifier's confidence or the likelihood of misclassification. The ℓ_p-norm is commonly used to quantify the difference between the original and perturbed inputs. Two primary approaches in adversarial attacks are *least-cost attacks* and *constrained attacks*, also referred to as *best-effort* and *bounded* attacks, respectively.

In *least-cost attacks*, the objective is to minimize the perturbation required to achieve a successful attack. This approach, originally formalized by Szegedy et al. [141], can be

expressed for an input $x \in \mathbb{R}^m$ targeting a specific class l as:

$$\text{minimize} \|r\|_2, \quad \text{subject to:} \quad F(x+r) = l, \quad x + r \in [0,1]^m$$

Here, r represents the perturbation, and $\|r\|_2$ is the ℓ_2-norm (Euclidean distance) of the perturbation, calculated as $\|r\|_2 = \sqrt{r_1^2 + r_2^2 + \cdots + r_m^2}$. The constraint $x + r \in [0,1]^m$ ensures that the perturbed input remains within valid limits (e.g., pixel intensities for images). The goal is to find the smallest perturbation r that still leads to misclassification into the target class l.

In contrast, *constrained attacks* operate under a fixed perturbation limit and aim to maximize the classifier's confidence in the adversarial class. Instead of minimizing perturbation, the goal is to increase the likelihood that the classifier assigns the target class l, without exceeding a predefined perturbation limit L. This can be formalized as:

$$\text{maximize} f_l(x+r), \quad \text{subject to:} \quad \|r\| \leq L, \quad x + r \in [0,1]^m$$

In this expression, $f_l(x+r)$ represents the classifier's confidence in class l, and L is the upper bound on the allowed perturbation r. The objective is to maximize the classifier's confidence in the adversarial class while ensuring that the perturbation does not exceed the given limit. Iterative optimization methods are often employed, with early stopping mechanisms when the perturbation exceeds the threshold.

In summary, *least-cost attacks* focus on minimizing perturbation while ensuring success, whereas *constrained attacks* aim to maximize classifier confidence within a fixed perturbation budget.

1.9 Summary

This chapter presents the foundational concepts necessary for understanding the integration of IoT, ML, and security in critical infrastructure systems, with a specific focus on WDNs. It also addresses the key security challenges arising from the increasing interconnection of IoT devices, highlighting the need for robust authentication and privacy measures. Additionally, the chapter explores the risks posed by AML and its potential to disrupt sensitive infrastructures. These concepts lay the groundwork for the further exploration of sustainable and secure infrastructure systems throughout the thesis.

Chapter 2

Revolutionizing Water Distribution Networks with IoT and ML

2.1 Introduction

Water Distribution Systems (WDSs) are critical infrastructures responsible for supplying potable water to urban, suburban, and rural populations. As global water demand increases and climate change exacerbates the complexities of water management, the need to optimize and modernize these systems has never been more urgent. Projections indicate that by 2030, the world could face a 40% water deficit [146], underscoring the importance of enhancing the efficiency, sustainability, and resilience of WDSs. To address these growing challenges, the integration of advanced technologies and innovative methodologies has become essential.

Water losses are a significant concern, with Europe alone losing an average of 26% of its water supply due to leaks, and some regions experiencing losses exceeding 50% [147]. Aging Water Distribution Networks (WDNs), which are a part of WDSs, further compound these inefficiencies, leading to substantial wastage. Climate change introduces additional strain, as it disrupts traditional water distribution patterns with more frequent and extreme weather events, fluctuating temperatures, and unpredictable precipitation cycles. These factors collectively contribute to increased pressure on aging infrastructure and heightened demand for water management solutions.

To tackle these pressing issues, the adoption of advanced Information and Communication Technology (ICT) solutions is critical for optimizing the monitoring, operation, and management of WDNs. In this context, Low Power Wide Area Networks (LPWANs), particularly those utilizing the LoRaWAN protocol, have emerged as a promising technology. LoRaWAN offers extensive coverage, low energy consumption, and long battery life, making it ideal for the deployment of large-scale, energy-efficient, and remote monitoring systems in WDNs. By leveraging these technologies, it becomes possible to create smart water distribution systems that are more responsive, efficient, and resilient to future challenges.

In this chapter, we explore approaches to enhance the intelligence and efficiency of WDNs. We begin by discussing network architectures, showing how hydraulic analysis can

be offloaded to the edge for localized, efficient computation. We then address the challenges of edge processing, providing a comprehensive overview of its implementation. To this end, we introduce two Long Range Wide Area Network (LoRaWAN) edge computing architectures and tools, both designed to integrate Machine Learning (ML) models for WDN analysis, prediction, and leak detection. We then conduct an in-depth analysis of gateway placement within a simulated LoRaWAN network integrated with a simulated WDN, emphasizing how network design influences overall system performance.

Following the discussion on network architecture, we present empirical results from deploying ML models, particularly neural networks (NNs), to predict hydraulic values. Our data-driven models, such as Graph Neural Networks (GNNs), alongside traditional approaches like Multilayer Perceptrons (MLPs) and convolutional neural networks (CNNs), demonstrate strong performance and generalizability.

Lastly, we analyze the impact of introducing random leakages to these models, discussing how their performance is affected by unexpected behavior and exploring potential improvements for more accurate leakage detection.

2.2 Smart Water Distribution Networks at the Edge

Traditional WDNs analyses typically rely on a centralized system where data is collected from End Devices (EDs) and processed in the cloud. While this approach is effective for basic operations, it encounters limitations in responsiveness and scalability as the network grows in size and data volume. To address these challenges, we propose a novel methodology that integrates edge computing within WDNs, transforming them into smarter, more efficient systems: Smart Water Distribution Networks (SWDNs). By utilizing the computational resources of network Gateways (GWs) as edge nodes, this approach enables localized execution of machine learning algorithms, facilitating real-time WDN data-driven analysis. This significantly reduces latency [148], allowing for faster decision-making, and ensures that critical water metering data, typically encrypted within the LoRaWAN protocol, is processed securely and efficiently at the edge.

The latest generation of LoRaWAN GWs is equipped to run advanced software, making ML analysis at the edge feasible. By processing data closer to the source, this approach not only reduces latency but also minimizes the volume of traffic on the backhaul network [149]. Furthermore, several system optimization strategies, including energy efficiency and faster response times, are achieved through parallel processing at the edge within the LoRaWAN framework.

The core of our idea is to train an optimal ML model using the aggregated data collected at the central server. Once trained, the model is deployed to the edge nodes, where it operates efficiently, making real-time inferences directly within the network.

Each node in the WDN is equipped with Internet of Things (IoT) metering devices to monitor water demand and other key metrics. These measurements feed into the ML mod-

els at the edge, enabling localized and responsive analysis that supports proactive network management.

2.2.1 LoRaWAN: The Optimal Choice for IoT-Based SWDNs

LoRaWAN and Narrowband Internet of Things (NB-IoT) dominate the IoT market, accounting for 84%, while Sigfox and LTE-M cover another 13% [150]. Although our solution is closely tied to LoRaWAN technology, it remains adaptable to broader fog-edge-cloud computing architectures and is compatible with various protocols. The primary goal is to harness the existing computational resources across the network infrastructure for precise, detailed analysis. A hierarchical computing structure enables diverse service offerings, improving resource management in smart grids. Additionally, advancements in 5G communication backbones are expected to boost fog computing by enhancing response times, reducing transmission delays, and lowering energy costs, particularly in delay-sensitive applications [151]. Emerging low-power network protocols, such as 6TiSCH, are also being explored to support efficient edge computing frameworks [152].

Figure 2.1 illustrates the architecture of a LoRaWAN-based IoT network, featuring a cell-free structure where IoT devices connect to multiple GWs based on their coverage areas. These GWs simultaneously receive data packets from IoT devices and forward appropriately demodulated packets to a central Network Server (NS). Our approach employs an IoT system equipped with water sensors strategically placed at key junctions within WDNs to collect essential measurements. These measurements are processed using ML algorithms at the network edge to visualize and predict hydraulic values.

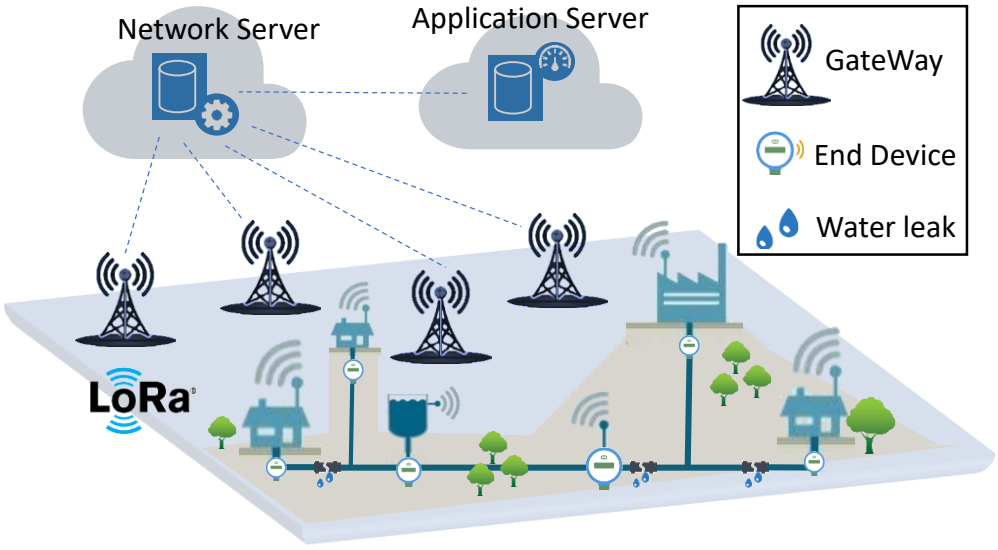


Figure 2.1: Architecture of a LoRaWAN-based IoT network. IoT devices connect to multiple gateways, which forward data to a central Network Server (NS) for processing.

As shown in Fig. 2.1, a WDN consists of interconnected links and nodes. The links represent pipes, while the nodes include junctions, tanks, or reservoirs. Junctions also serve

as demand points within the network, where water usage is monitored. Tanks provide finite water sources, whereas reservoirs offer an infinite supply of hydraulic resources.

IoT devices with low power consumption and long-range communication are particularly well-suited for such applications. WiFi-based networks, cellular networks, and LPWANs represent the three primary trends in IoT communication, with LPWANs expected to dominate [153].

As depicted in Fig. 2.1, the LoRaWAN architecture adopts a star-of-stars topology, where EDs communicate with one or more GWs, which forward data to the NS and ultimately to the IoT application server. In our scenario, LoRaWAN EDs are deployed at WDNs junctions to collect water demand data.

2.2.2 Optimizing Gateway Placement for Efficient Edge Processing in SWDNs

We emphasize the importance of gateway (GW) positioning in SWDNs. As part of our work, we demonstrate a practical example of GW deployment for edge processing using customized WDNs, inspired by public networks from EPANET [154].

To evaluate the performance of LoRa cells in SWDNs, we developed a custom-built simulation tool based on the publicly available LoRaSim simulator [155]. This tool, named LoRaSURFING, is a discrete event simulator implemented in Python. In our simulations, N LoRa EDs are distributed across a two-dimensional space, either in a grid layout or randomly. The simulation involves M GWs, which serve as data collection points for the network.

Table 2.1: Radio simulation parameters.

Parameter	**Value**
Carrier Frequency (MHz)	863.0
Bandwidth (kHz)	125
TX Power	14 dBm
Path loss values	$\eta = 2.9$, $\sigma^2 = 0$, $\overline{L_{pl}}(40m) = -66\ dB$

The communication quality between each ED and its corresponding GW is evaluated using the Received Signal Strength Indicator (RSSI), modeled based on the distance between the devices and following a path loss model. The parameters used in the simulations are summarized in Table 2.1, where $\overline{L_{pl}}(d_0)$ denotes the mean path loss at a reference distance of $d_0 = 40\,m$.

For network planning, we integrate the Water Network Tool for Resilience (WNTR) tool into the LoRaSim simulation environment, allowing us to focus on custom-designed WDNs that reflect real-world conditions.

The key parameters for the WDNs used in our simulation experiments are detailed in Table 2.2. We simulate various LoRaWAN network scenarios using LoRaSIM, exploring three sub-networks, labeled ”A,” ”B,” and ”C,” each with increasing complexity in terms of

Table 2.2: Parameters of the WDNs used to test the proposed approach.

WDN	No. of Nodes	No. of Reservoirs	No. of GWs
A	83	1	4
B	1038	1	51
C	2099	2	84

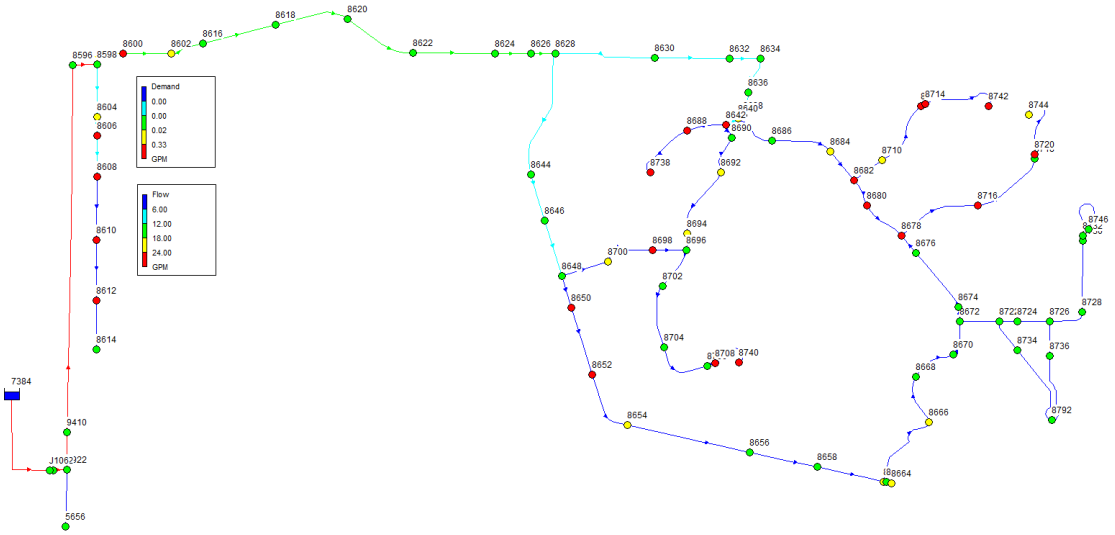
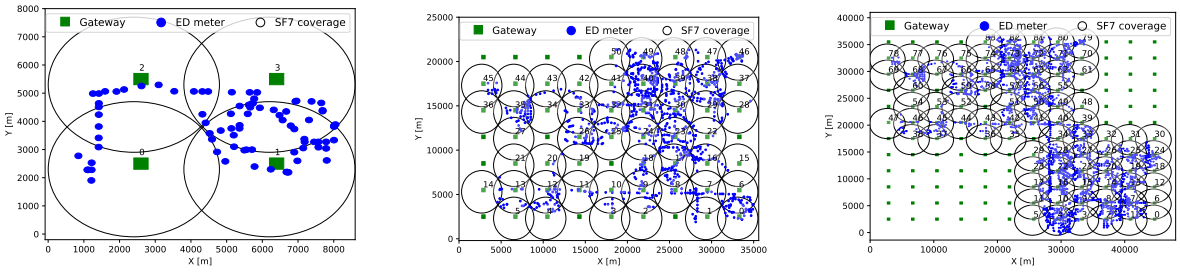


Figure 2.2: Example of WDN branch with reservoir, nodes and flow measurements.

node count and interconnections. Figure 2.2 illustrates Network "A," where two color scales depict node demand (circle marker color) and pipe flow (connecting segment color).



(a) Network A (b) Network B (c) Network C

Figure 2.3: Topology of the considered networks: (a) Network A, (b) Network B, and (c) Network C.

Table 2.2 also summarizes the datasets generated for each network. Specifically, the smallest dataset (Figure 2.3a) is generated from a sub-network with $n_1 = 83$ junctions, $m_1 = 1$ reservoir, and $x_1 = 0$ tanks. The medium-sized dataset (Figure 2.3b) comes from a sub-network with $n_2 = 1038$ junctions, $m_2 = 1$ reservoir, and $x_2 = 0$ tanks. The largest dataset (Figure 2.3c) is derived from a sub-network with $n_3 = 2099$ junctions, $m_3 = 2$ reservoirs, and $x_3 = 0$ tanks.

The network topologies considered in this study are depicted in Figure 2.3, where nodes are shown as blue circles and LoRaWAN GWs as green squares, with their coverage areas represented as circles.

These figures highlight the critical role of optimal GW placement. Each GW can only collect data from nodes within its coverage area, and misplacement can lead to incomplete or inaccurate measurements. The inference process considers only those gateways with nodes inside their coverage areas, making their correct positioning essential. The indexes on the y-axis of Figure 2.3 reflect the gateways' positions in the simulation. Proper GW placement is crucial, as it directly impacts network performance and ensures accurate data collection.

2.3 Enhanced Stream Processing in WDNs

In real-world WDNs, vast amounts of data are generated, classifying these systems as *Big Data* systems. Traditional cloud-based solutions often experience bottlenecks and struggle to fully exploit IoT capabilities, while Fog-based solutions may lack sufficient resources for effective Big Data processing.

To overcome these limitations, distributed approaches are emerging, where computational tasks are strategically balanced by offloading lighter processes to the network edge, while more complex computations are handled in the cloud. This approach is achieved through function mapping, where local data is aggregated and only essential results are transmitted upstream. Frameworks such as Hadoop 2.0 [156], Spark [157], and Flink [158] are designed to optimize these distributed processing workflows.

NebulaStream [159], a robust data processing platform, addresses critical issues such as heterogeneity, unreliability, and scalability in IoT systems. It is an IoT stream-processing framework that enables efficient edge processing [149].

Building on this, we propose LEAKSTREAM [160], a solution that integrates stream processing with LPWAN-based networks to enhance leak detection, monitoring, and prediction in WDNs. LEAKSTREAM utilizes IoT water sensors, typically installed at consumer utility points, to monitor water usage. These data are processed at the network edge through ML algorithms.

Our distributed approach deploys ML algorithms at the edge, using k-means clustering and ML models to analyze WDN data. The method further partitions the network into zones using clustering techniques, facilitating the processing of measured data and enabling the training of ML models that generalize across the WDN structure.

Instead of relying on flow sensors along the pipes to measure internal hydraulic flows, our approach focuses on analyzing WDNs using data from utility meters. By monitoring hydraulic flows at utility sources (demand points at junctions), our method aligns with the increasing adoption of smart meters in utility services [161].

This framework builds on prior work [162, 163] that suggests combining nodes with similar leakage behaviors to improve classification accuracy. Our goal is to simplify the architecture, reduces the number of features, and incorporate stream processing for real-time performance.

2.3.1 Scalable Wireless Network Solutions

LPWANs offer significant potential but face scalability challenges in large-scale deployments. We already said that LoRaWAN addresses scalability issues by enabling real-time data processing at the network edge. This approach improves system performance and optimizes resource allocation, making it well-suited for large-scale applications.

To further reduce latency and minimize network traffic, we leverage the NebulaStream framework. As shown in Fig. 2.1, GWs under the control of the NS are responsible for local data processing, while the NS coordinates remote operations efficiently.

Our method centers on developing ML models designed to capture the hydraulic behavior within specific network zones. The process starts by clustering network nodes based on their spatial proximity and observed trends in pressure and demand values. These features offer the possibility to be suitable to to train any kind of ML models.

2.3.2 Stream Processing in SWDNs Edge Frameworks

Unlike traditional cloud-based systems, where data is centrally collected and processed in batches, we propose a sensor-fog-cloud architecture that dynamically allocates data stream processing across the network, from sensors at the source to the cloud.

In this section, we detail the implementation of the LEAKSTREAM algorithm using this sensor-fog-cloud model within the *NebulaStream* platform.

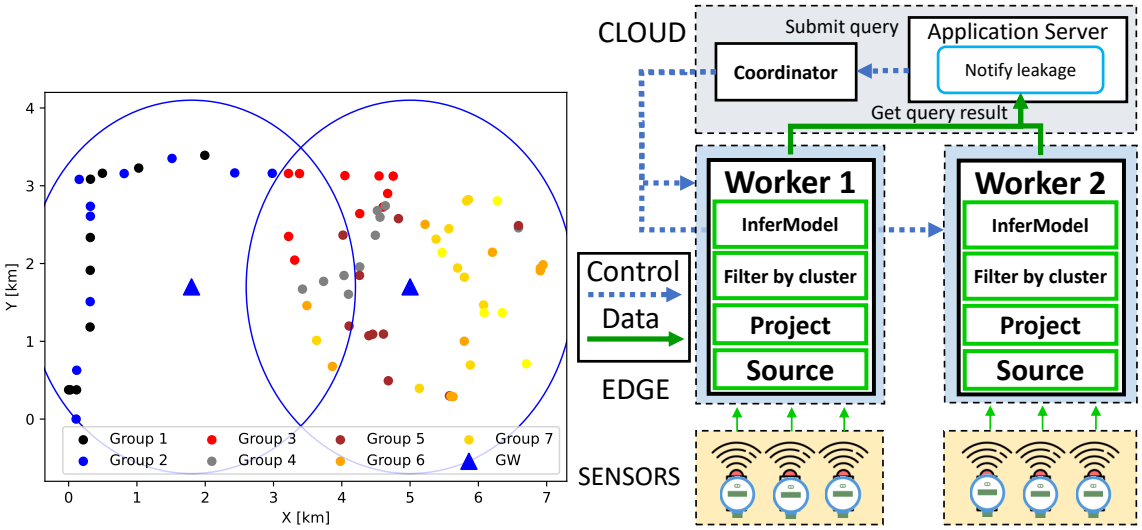
LEAKSTREAM operates in two stages: i) k-means clustering groups WDN junctions into zones based on similar characteristics, and ii) eventually, a ML model performs inference in these zones by analyzing the dynamic flow of the network. This approach can be used to study various behaviors, such as regular pressure predictions or leak classification.

Based on the selected network topology, we determine optimal positions for the GWs responsible for collecting monitoring data. Fig. 2.4a illustrates the placement of the GWs (shown as triangular markers) and their coverage areas (depicted by large blue circles). Each GW covers a subset of the total WDN measurements within its area.

NebulaStream facilitates distributed query execution through workers operating at the edge. As shown in Fig. 2.4b, these workers manage data flow hierarchically from the source to the sink, handling inputs from various sources, applying specific operators, and forwarding results to the next worker or to the data sink. A cloud-based coordinator manages external queries through an interface.

In the LoRaWAN network, the GWs act as the edge layer, equipped with proxy agents to stream data into the system. To ensure secure data flow, the LoRaWAN protocol's activation by personalization method is configured to access encrypted information from the monitored network.

Fig. 2.4b shows the structure of our implementation, which includes two workers—one for each GW. The diagram outlines the operators used for data pre-processing and support of the LEAKSTREAM algorithm. At the edge layer, the *source* operator initiates the data



(a) Nodes clusters (b) Stream processing implementation

Figure 2.4: Map of the nodes and GWs position, nodes are coloured according to the selected clustering, GWs are provided together with the coverage area (a). Example of deployed operators that implement LEAKSTREAM on GWs workers.

stream, followed by the *projection* operator, which selects relevant data fields. The *filter* operator extracts data associated with specific clusters, and the infer model operator runs inference on the incoming data using a pre-trained ML model. At the cloud layer, ML results from the network clusters are aggregated and forwarded to the application server. In this specific case we show how to aggregate data for leakage detection which triggers the leakage alarm if necessary.

To illustrate the query execution, we provide a Java-based example that summarizes the data processing steps within NebulaStream.

```
// Provide input parameter
String CLUSTER = ARG_PAR_1
List FEATURES_LIST = ARG_PAR_2
//create stream and select subset of fields
Query stream = new Query().from("WDN_STREAM")
stream.from(gatewaystream).project(FEATURES_LIST)
    .filter(CLUSTER)
    .inferModel(Objects.requireNonNull(
        InferModel.class.getResourceAsStream("/model.tflite")))
       .on(attribute(FEATURES_LIST)
       .as(attribute("leakage_prediction", BasicType.FLOAT32));
    .sink(new MQTTSink());
```

We show our framework on the demo WDNs discussed in the last section. Fig. 2.4a displays the 2D positions of nodes clustered by the k-means algorithm, where nodes within the same cluster are shown in the same color. To determine the optimal number of clusters, we apply the Elbow and Silhouette Methods, which empirically assess inter- and intra-cluster

similarities. After testing various k values, we identify $k = 7$ as the optimal cluster number. It is important to note that clusters are formed based on average demand values and pressure rather than physical proximity, explaining why nodes in the same cluster may not be spatially close in Fig. 2.4a.

This work demonstrated the potential feasibility of a stream processing LoRaWAN network for specific WDNs and outlined how the entire flow of operations and information could be structured.

2.4 LoRaWAN Deployment and Gateway Distribution Analysis

We previously explored how SWDNs integrate seamlessly with LoRaWAN networks, where IoT EDs function as meters within the physical WDN. LoRaWAN operates on a star-of-star topology, enabling single-hop communication between IoT EDs and GWs, which relay data to the application server via an IP network [164, 12]. The architecture comprises three key components: the network server, one or more GWs, and IoT EDs [165]. While supporting large-scale deployments with minimal complexity, the system is limited by throughput constraints [166].

As discussed, optimal gateway positioning and network planning are crucial for robust system performance. Poor planning can result in connectivity issues [167]. Although adding more gateways improves coverage, it also raises CAPEX and OPEX [168, 169], making it essential to balance Quality of Service (QoS) and costs through strategic gateway placement [170].

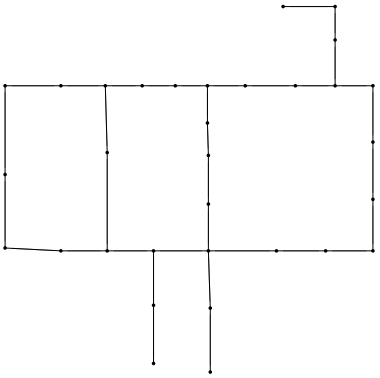
Two key parameters in LoRaWAN are the Spreading Factor (SF) and Data Extraction Rate (DER). SF mediates the trade-off between data rate and range: higher SF extends range but lowers data rate, while lower SF increases data rate but reduces range. DER reflects communication reliability, with higher values indicating more successful packet reception at the network server.

2.4.1 Using more standardized WDNs

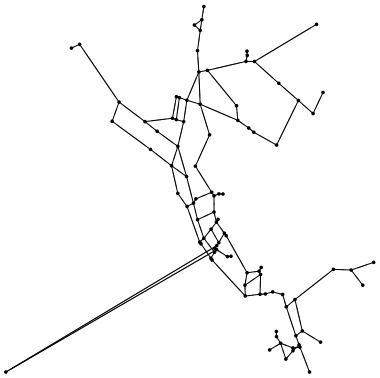
Network	Junctions	Tanks	Reservoirs	Pipes	Pumps	Valves
Hanoi	31	0	1	34	0	0
Net3	92	3	2	117	2	0
C-Town	388	7	1	429	11	4

Table 2.3: Node and Link Distribution for Different Water Networks

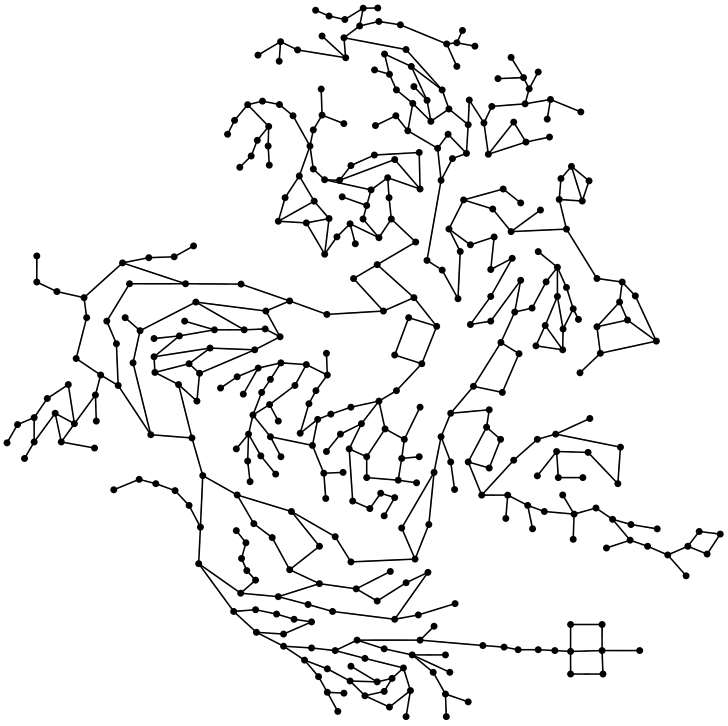
In this section, we shift from the previous custom-shaped WDNs to standardized, widely-referenced networks, commonly used in hydraulic analysis. This transition establishes a foundation for the ML hydraulic studies and implementations discussed in Section 2.5 and



(a) Hanoi WDN.



(b) Net3 WDN.



(c) C-Town WDN.

Figure 2.5: All the WDNs simulated and employed in our analysis.

Chapter 3. We focus on three well-known WDNs: Hanoi (Figure 2.5a), Net3 (Figure 2.5b), and C-Town (Figure 2.5c). These networks are publicly available and frequently used in simulation benchmarks, such as WNTR and EPANET, as well as in scientific literature. They offer diverse node and link configurations, making them suitable for various analyses.

The Hanoi network consists of 32 nodes and 34 links, Net3 includes 97 nodes and 119 links, and C-Town, the largest, comprises 396 nodes and 444 links (Table 2.3). The nodes include junctions, tanks, and reservoirs, while the links consist of pipes, pumps, and valves.

2.4.2 Methodology and Results for DER Analysis

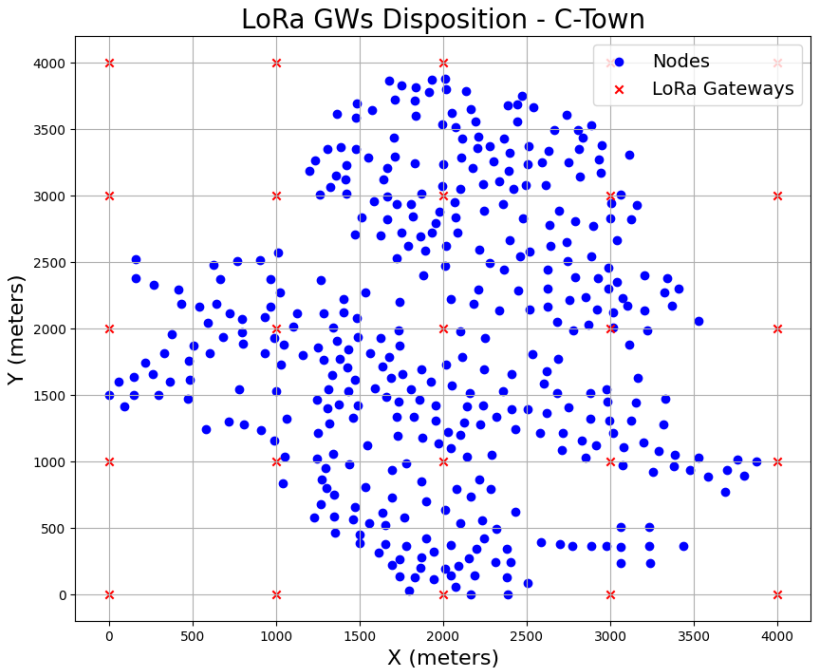


Figure 2.6: Spatial distribution of LoRa GWs in the simulated C-Town WDN.

Figure 2.6 illustrates the spatial distribution of gateways in the C-Town simulated WDN, where 25 GWs were deployed. Similarly, Figures 2.7 and 2.8 show the gateway distribution in the simulated Hanoi and Net3 WDNs, where 25 and 1 GWs were deployed, respectively. Only one gateway is needed for Net3, as it covers a small, concentrated area, making a single gateway sufficient for efficient coverage.

We analyzed DER in this and the ”Hanoi” and ”Net3” WDNs, where the number of IoT devices matched the nodes of each WDN. Simulations were conducted using ns3, with variations in both sampling frequency and SF to assess performance.

Despite being simulations, these scenarios offer useful insights for real-world applications. Net3 represents a dense, crowded environment (e.g., an office), Hanoi reflects a sparse, wide-area network, and C-Town falls between the two, offering a balanced deployment. These differences highlight how network characteristics influence performance and can guide real-world deployment strategies.

Figures 2.9, 2.10, and 2.11 display DER versus application period for different SFs in

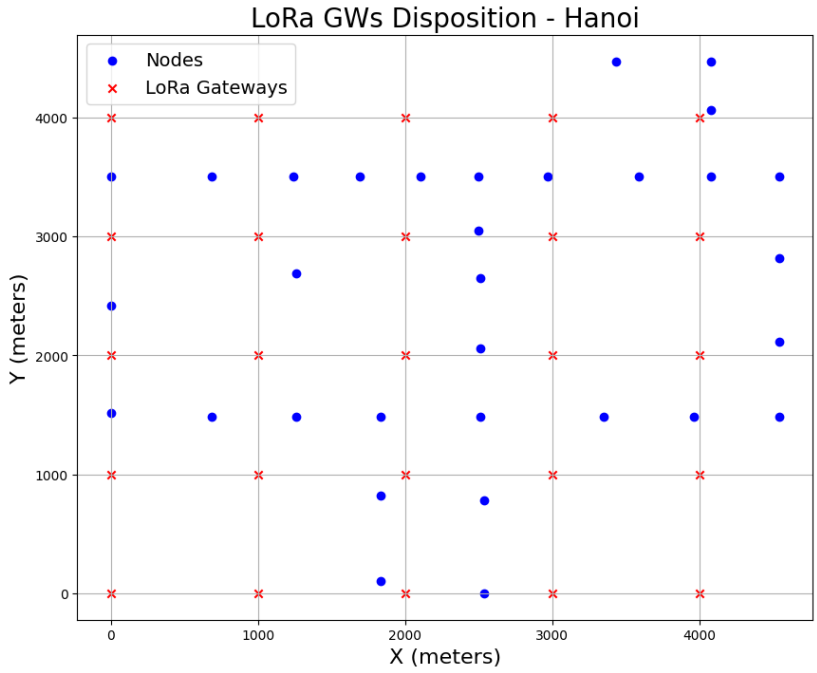


Figure 2.7: Spatial distribution of LoRa GWs in the simulated Hanoi WDN.

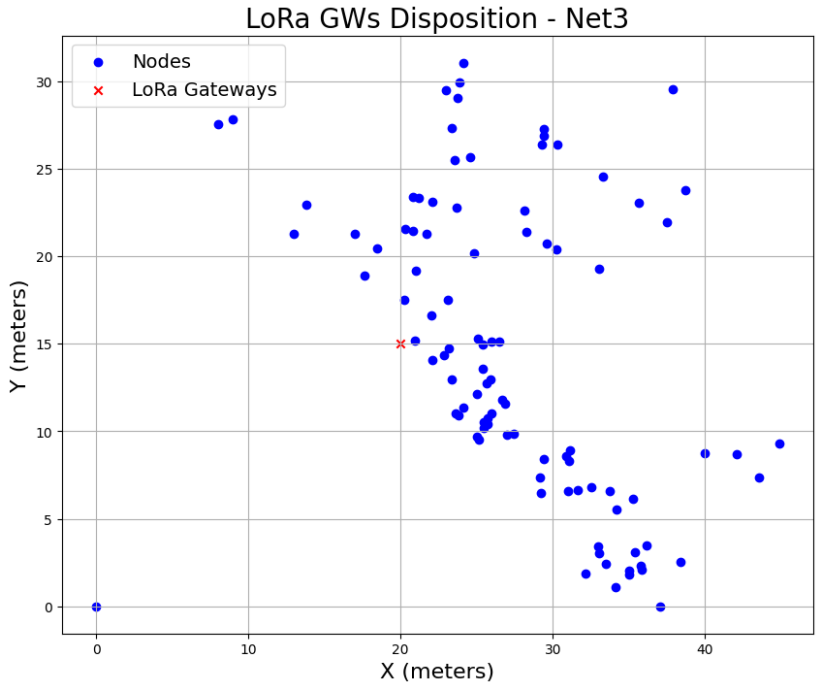


Figure 2.8: Spatial distribution of LoRa GWs in the simulated Net3 WDN.

C-Town, Hanoi, and Net3. The application period refers to the interval at which an application running on top of a LoRaWAN node generates and transmits packets. In C-Town and Net3, higher SFs (e.g., SF12) result in lower DER for shorter application periods, but DER improves as the application period increases. This suggests that longer intervals between data transmissions can enhance reliability for long-range applications. In contrast, Hanoi shows consistently high DER across all SFs, indicating that well-planned, well-covered networks can maintain high reliability even with frequent data transmissions.

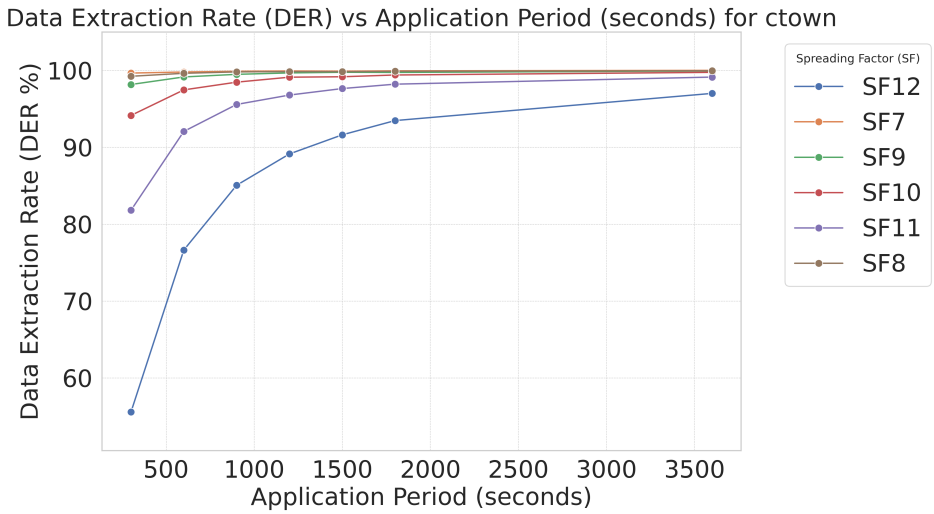


Figure 2.9: Data Extraction Rate (DER) vs Application Period (seconds) for different Spreading Factors (SF) in the simulated C-Town Water Distribution Network.

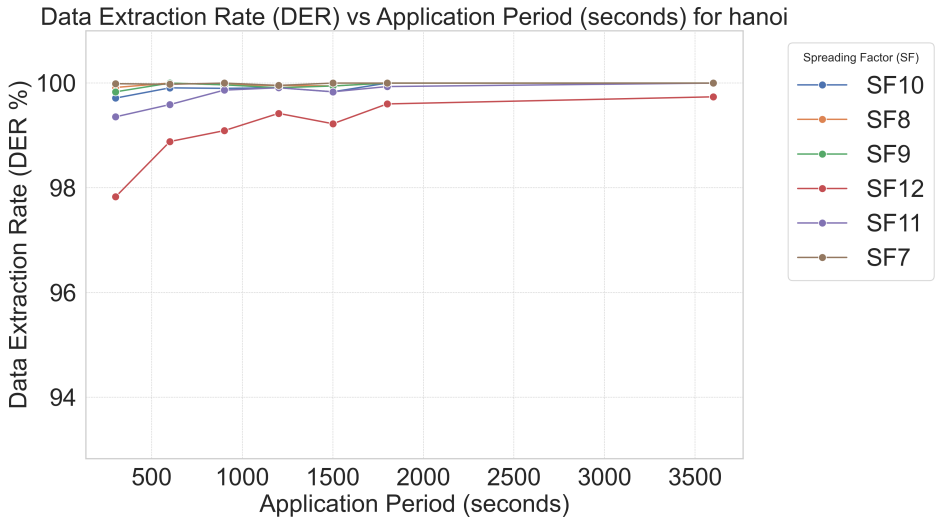


Figure 2.10: Data Extraction Rate (DER) vs Application Period (seconds) for different Spreading Factors (SF) in the simulated Hanoi Water Distribution Network.

In all scenarios, carefully balancing SF and application period is crucial for optimizing DER, especially in dense or interference-prone environments. Although these are simulations, the insights gained are highly valuable for real-world applications. These results provide a strong foundation for understanding the potential challenges and optimizations required for future real-world deployments of SWIM.

2.5 Integrating Neural Networks into a SWDN

We have showed that SWDNs can provide a continuous stream of data from IoT devices, such as water demand measurements at network nodes and other key hydraulic metrics. Furthermore, we discussed how a ML model can be deployed at the network edge to perform WDN analysis directly at that level. The primary objective of the edge computing architectures pre-

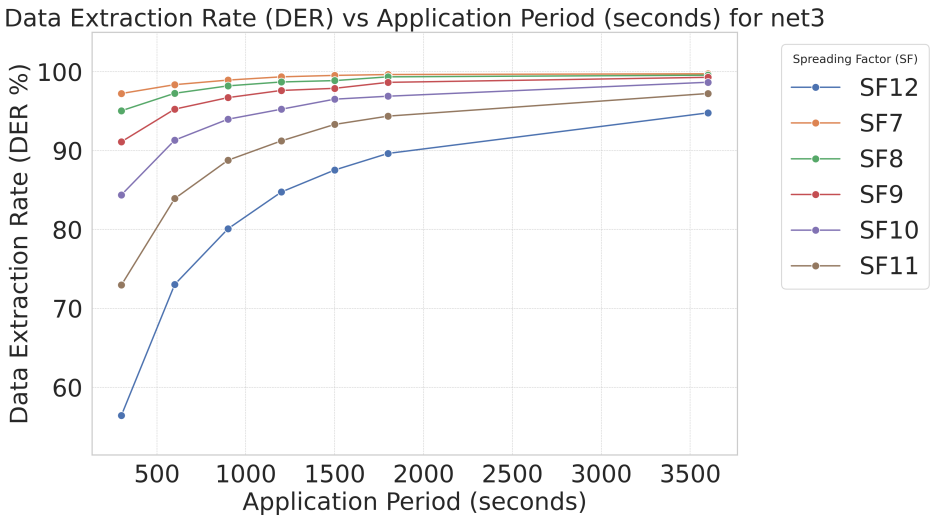


Figure 2.11: Data Extraction Rate (DER) vs Application Period (seconds) for different Spreading Factors (SF) in the simulated Net3 Water Distribution Network.

sented in Sections 2.2 and 2.3 is to leverage ML models within these critical systems, specifically designed for this purpose. Current research, such as [171], focuses on developing ML models that perform reliably under various uncertainties. Once trained, these models provide significant efficiency gains over traditional hydraulic simulations, which often struggle with computational constraints and slower response times, particularly in real-time or large-scale environments [172].

A data-driven ML model can be trained and continuously fine-tuned using this data, allowing dynamic adaptation to real-time changes in the network. In our work, we simulate this scenario by training and evaluating on hydraulic data generated using the EPANET and WNTR simulation tools, which provide realistic conditions for creating comprehensive datasets.

One of the core tasks in this framework is predicting pressure values at network junctions using a minimal set of features typically derived from EPANET configuration files. To address this, we explore several neural network architectures. These include a standard MLP, chosen for its simplicity, a 1D CNN designed to capture sequential patterns in the data, and a Graph Convolutional Neural Network (GCNN) with GINEConv layers, which uniquely incorporate both node and edge features (e.g., pipes) to enhance the model's representation of the network structure.

A distinctive aspect of our models is their ability to generalize complex hydraulic calculations and derive positional information from a minimal set of features— in the case of non-GNN models, even from just a single feature per Junction. This capability is crucial as it allows us to significantly enhance the efficiency of the system by relying on a highly reduced dataset.

In our current results, the ML models are trained on the full composition of the network, using extreme scenarios where random demand patterns are assigned to each junction for every sample. This approach ensures the models generalize well for pressure predictions

without being biased by repetitive demand patterns. Consequently, the trained models exhibit strong robustness and adaptability when applied to real-world datasets, which tend to feature more regular patterns. This adaptability enables further applications such as pressure prediction, leak detection, and comprehensive water network analysis.

2.5.1 Hardware Setup

Our hardware setup for generating the datasets, training and testing the ML models, and performing all processing tasks consists of a Linux computer running Linux Mint 22, with Linux kernel 6.8.0-45, an Intel I9-9900KF CPU, 64GB of RAM, and an RTX 2080 Ti GPU. We use PyTorch as the ML framework for NNs implementations. To ensure reproducibility, we set fixed random seeds for all experiments, and we plan to publicly release the code in the near future.

2.5.2 Latin Hypercube Sampling for Dataset Generation

For dataset generation, we utilize the Hanoi, Net3, and C-Town WDNs, as discussed in Section 2.4.1. To simulate a wide variety of demand patterns across the junctions, we employ *Latin Hypercube Sampling (LHS)*, an efficient statistical technique for sampling from a multidimensional distribution. This method ensures a broad exploration of the demand space by systematically generating diverse combinations of demand values for each junction.

Let n_{nodes} represent the number of junctions in the network, and $n_{\text{params}} = n_{\text{nodes}}$. Using LHS, we generate $n_{\text{samples}} = 20,000$ unique demand scenarios, each corresponding to a distinct combination of demand values across all nodes. This results in a matrix $\mathbf{LHD} \in \mathbb{R}^{20000 \times n_{\text{params}}}$, where each element $\mathbf{LHD}[i, j]$ is a random sample from a uniform distribution over the range $[0, 1]$.

LHS divides the distribution of each parameter into 20,000 equal-probability intervals, ensuring comprehensive coverage of the parameter space. For each parameter j, a value x_{ij} is sampled from the i-th interval as:

$$x_{ij} \in \left[\frac{i-1}{20000}, \frac{i}{20000}\right], \quad \text{for } i = 1, 2, \ldots, 20000$$

These sampled values are then shuffled across parameters to create distinct scenarios for each combination of demand values.

For each node j, the demand pattern in scenario i is obtained by scaling $\mathbf{LHD}[i, j]$ to a predefined range $[d_{\text{min}}, d_{\text{max}}]$, which is set to $[0.1, 2.0]$. This scaling is computed as:

$$d_{ij} = d_{\text{min}} + \mathbf{LHD}[i, j] \cdot (d_{\text{max}} - d_{\text{min}}) = 0.1 + \mathbf{LHD}[i, j] \cdot 1.9$$

where d_{ij} is the demand at node j in scenario i, and $\mathbf{LHD}[i, j]$ is the corresponding LHS sample. This ensures a diverse range of demand patterns, capturing various operating conditions of the network.

We input these demand patterns into the WNTR simulator and execute 20,000 simulations, saving all generated results. By systematically generating distinct demand patterns for each junction, this approach provides a comprehensive dataset for evaluating WDNs performance under diverse configurations and conditions.

2.5.3 Neural Network Architectures for Pressure Prediction

In this work, we implement three distinct neural network architectures to predict pressure values at the junctions of WDNs. Each model is designed to address the regression task of estimating the pressure at every junction based on network features. Below is a description of each architecture and its components.

Input Features

The sole node-derived input feature for all models is the expected demand for each junction node. This expected demand is calculated by combining base demands with demand multipliers, generated using LHS for every junction and time step.

For the GNN, we extend the input to include edge features, as GNNs can leverage the interconnected relationships between nodes. Specifically, the diameter and length of each pipe (link) are used as edge features.

SimpleMLP

The first architecture is a simple feedforward neural network, or MLP, designed to predict pressure values for each junction. The model consists of three fully connected layers, using the LeakyReLU activation function to introduce non-linearity. The input corresponds to the set of features for all junctions in the WDN, and the output provides the pressure predictions for all junctions ($n_{\text{junctions}}$) in the network. The architecture for a network with $n_{\text{junctions}}$ is structured as follows:

$$\text{SimpleMLP:} \begin{cases} \text{fc1}: & \text{Linear}(n_{\text{junctions}} \to 128) \\ \text{fc2}: & \text{Linear}(128 \to 128) \\ \text{fc3}: & \text{Linear}(128 \to n_{\text{junctions}}) \\ \text{Activation}: & \text{LeakyReLU}(slope = 0.01) \end{cases}$$

This MLP is trained using mean-squared error (MSE) as the loss function, with a learning rate of 0.001 for 100 epochs. We save the best model based on validation loss performance.

Conv1DCNN

The second architecture is a 1D CNN which incorporates two 1D convolutional layers followed by fully connected layers, with LeakyReLU activations. Additionally, a dropout layer

is used to prevent overfitting. Similar to the MLP, the output corresponds to the number of junctions ($n_{\text{junctions}}$) in the network. The architecture for a network with $n_{\text{junctions}}$ is as follows:

$$\text{Conv1DCNN:} \begin{cases} \text{conv1}: & \text{Conv1d}(1 \to 64, \text{kernel} = 3, \text{stride} = 1, \text{padding} = 1) \\ \text{conv2}: & \text{Conv1d}(64 \to 32, \text{kernel} = 3, \text{stride} = 1, \text{padding} = 1) \\ \text{fc1}: & \text{Linear}(992 \to 128) \\ \text{fc2}: & \text{Linear}(128 \to n_{\text{junctions}}) \\ \text{Dropout}: & p = 0.2 \\ \text{Activation}: & \text{LeakyReLU}(slope = 0.01) \end{cases}$$

The training setup is identical to the MLP, with MSE as the loss function, a learning rate of 0.001, and early stopping based on the best validation loss.

GINENet

The third architecture is a GNN that relies on Graph Isomorphism Network with Edge features (GINEConv) layers, specifically designed for node-level prediction tasks. In this case, each junction is treated as a node in a graph, with the connections (pipes) between them as edges. The GNN processes both node and edge features, leveraging the GINEConv layers to better capture the topological structure of the network. The architecture is as follows:

$$\text{GINENet:} \begin{cases} \text{conv1}: & \text{GINEConv}(\text{nn} = \text{Linear}(1 \to 256)) \\ \text{conv2}: & \text{GINEConv}(\text{nn} = \text{Linear}(256 \to 256)) \\ \text{conv3}: & \text{GINEConv}(\text{nn} = \text{Linear}(256 \to 1)) \\ \text{Activation}: & \text{LeakyReLU}(slope = 0.01) \end{cases}$$

The GNN is trained under the same conditions as the MLP and 1D CNN, with the regression task aimed at predicting pressure at each junction (treated as a node in the graph).

Training Setup and Evaluation

All three models are trained with a learning rate of 0.001, using MSE as the loss function. The training is conducted over 100 epochs, with the model showing the lowest validation loss being selected as the best-performing model. The dataset is randomly split into 70% training, 15% validation, and 15% test sets, with both the input features and labels normalized using a standard scaler to ensure consistent training.

The dataset structure and processing vary depending on the model architecture:

- SimpleMLP: The input data is flattened into a 2D matrix of shape `(batch_size, num_features)`, where `num_features` is the product of the number of nodes and the

features per node. This format treats the data as independent vectors, disregarding spatial or relational information.

- Conv1DCNN: The input is structured as a 3D tensor with shape `(batch_size, num_nodes, num_features)`, where `num_nodes` represents the distinct spatial locations, and `num_features` corresponds to the attributes per node.
- GINENet: The input consists of graph-structured data, where each sample is a graph object. The node features are represented as `(num_nodes, num_node_features)`, while the edge features and adjacency information are stored as `(num_edges, num_edge_features)` and a connectivity matrix, respectively.

In addition to MSE, two supplementary evaluation metrics—Mean Absolute Error (MAE) and the coefficient of determination (R^2)—are tracked for the validation and test sets, providing further insights into model performance and generalization.

Task and Output

Both the MLP and 1D-CNN architectures are tasked with predicting a single pressure value for each junction in the WDN. The GNN, designed for node-level regression, similarly predicts one pressure value per junction but incorporates additional structural information. Despite differences in architecture, all models produce an output for every junction in the network, with the number of outputs corresponding to the total number of junctions in each specific WDN.

2.5.4 Training and Evaluation Results

Neural Network	WDN	Best Epoch	Train Loss	Train MSE	Train MAE
SimpleMLP	Hanoi	95	0.0003	0.0003	0.0118
	Net3	89	0.0003	0.0003	0.0136
	C-Town	81	0.0070	0.0070	0.0587
Conv1DCNN	Hanoi	68	0.0142	0.0142	0.0885
	Net3	95	0.0173	0.0173	0.0982
	C-Town	91	0.1762	0.1762	0.3100
GINENet	Hanoi	46	0.5011	0.5011	0.5392
	Net3	95	0.2069	0.2069	0.1113
	C-Town	80	0.1456	0.1456	0.1395

Table 2.4: Training Metrics for Different Neural Networks and Datasets at Best Validation Loss Epoch

The training and validation metrics for all models are summarized in Tables 2.4 and 2.5.

For the Hanoi network, the SimpleMLP achieves the highest performance, with a validation MSE as low as 0.0001 and an R^2 score approaching 1.0, indicating near-perfect prediction accuracy. The Conv1DCNN also performs well, though its validation metrics show a slight decrease in accuracy compared to the MLP. In contrast, the GNN shows significantly

NN	WDN	Best Epoch	Val Loss	Val MSE	Val MAE	Val R^2
SimpleMLP	Hanoi	95	0.0001	0.0001	0.0075	0.9999
	Net3	89	0.0002	0.0002	0.0086	0.9998
	C-Town	81	0.0063	0.0063	0.0532	0.9937
Conv1DCNN	Hanoi	68	0.0019	0.0019	0.0343	0.9981
	Net3	95	0.0024	0.0024	0.0384	0.9977
	C-Town	91	0.0919	0.0919	0.2172	0.9085
GINENet	Hanoi	46	0.4916	0.4916	0.5326	0.5038
	Net3	95	0.2056	0.2056	0.1118	0.7944
	C-Town	80	0.1454	0.1454	0.1380	0.8545

Table 2.5: Validation Metrics for Different Neural Networks and Datasets at Best Validation Loss Epoch

lower performance on this smaller dataset, with an R^2 around 0.5, highlighting the advantage of simpler models for smaller networks like Hanoi.

In the case of the Net3 network, the Conv1DCNN achieves the best overall performance, obtaining the lowest validation MSE and the highest R^2 score among all models. The SimpleMLP also performs strongly, with results close to those of the Conv1DCNN. However, the GNN, while performing better on this larger network than on Hanoi, still lags behind the other models in terms of accuracy.

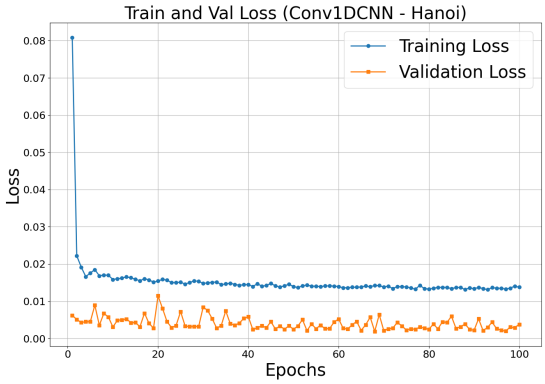
For the C-Town network, which is significantly larger and more complex, the SimpleMLP once again delivers the best results, with an R^2 of 0.9937 and a low validation error. The Conv1DCNN exhibits a slight drop in performance, with a validation MSE of approximately 0.092, while the GNN demonstrates reasonable accuracy, achieving an R^2 of 0.8545. This suggests that the GNN is better suited to handling the complexity of larger networks compared to smaller-scale cases.

Across all networks, the SimpleMLP consistently provides the best results, especially in smaller networks like Hanoi, where its simplicity proves advantageous. The Conv1DCNN remains competitive, particularly in the Net3 network, where it slightly outperforms the MLP. The GNN, while better suited to larger networks like C-Town, struggles on smaller datasets, where simpler architectures like the MLP and CNN excel.

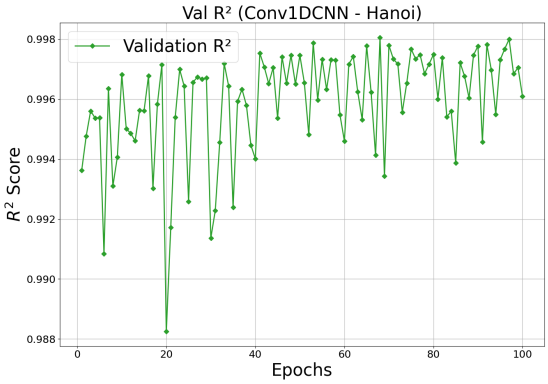
Test Performance Results

The models are also evaluated on the test set using the best weights saved during training. The results, summarized in Table 2.6, reveal the performance differences between the three neural network architectures across the Hanoi, Net3, and C-Town datasets. Test metrics are reported in the unnormalized domain, ensuring the values are interpretable in real-world terms, reflecting actual pressure values within the WDNs.

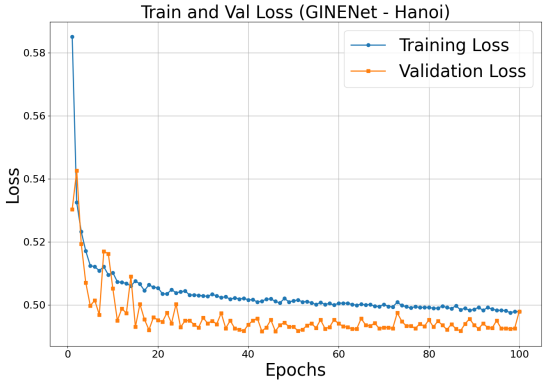
The SimpleMLP consistently achieved the best test performance on smaller networks, such as Hanoi and Net3, with near-perfect accuracy, evidenced by a test R^2 of 0.9999 for both networks and extremely low MSE and MAE values. These results confirm the MLP's strong ability to generalize in less complex networks, where its simplicity proves advantageous.



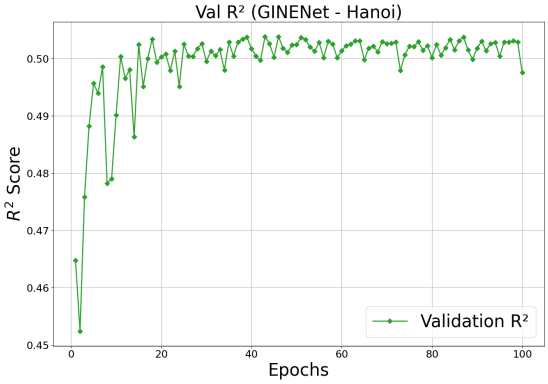
(a) Conv1D Loss Curves - Hanoi



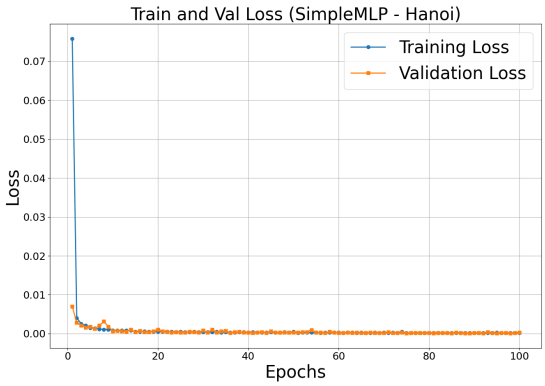
(b) Conv1D R^2 Scores - Hanoi



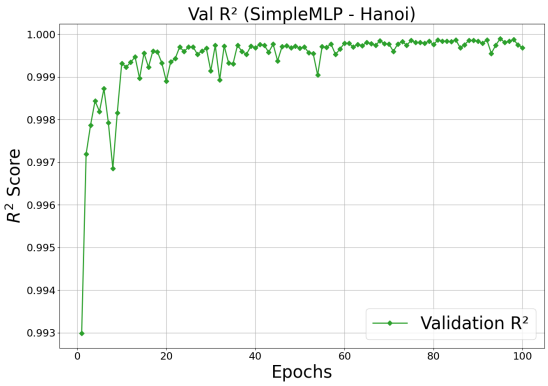
(c) GINENet Loss Curves - Hanoi



(d) GINENet R^2 Scores - Hanoi

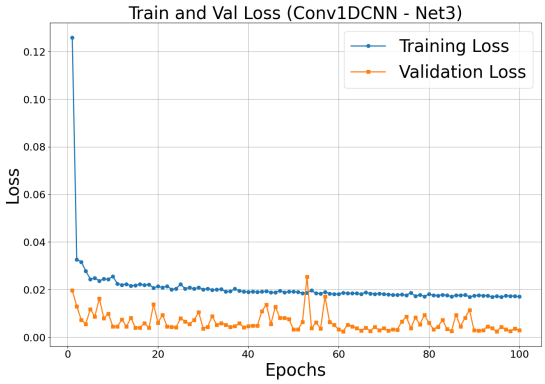


(e) MLP Loss Curves - Hanoi

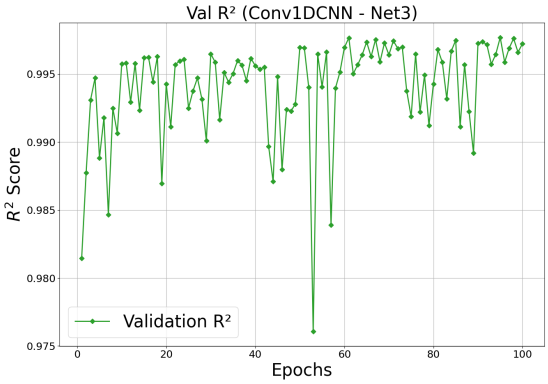


(f) MLP R^2 Scores - Hanoi

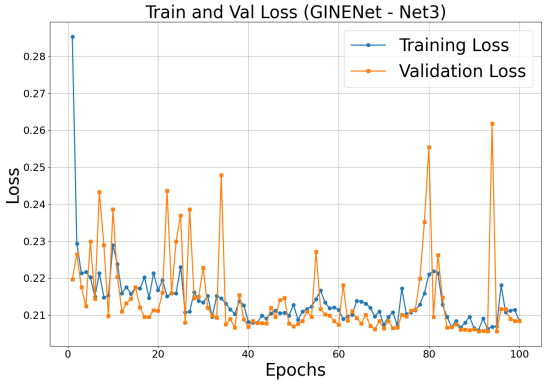
Figure 2.12: Training Loss Curves and R^2 Scores for Conv1D, GINENet, and MLP on the Hanoi dataset.



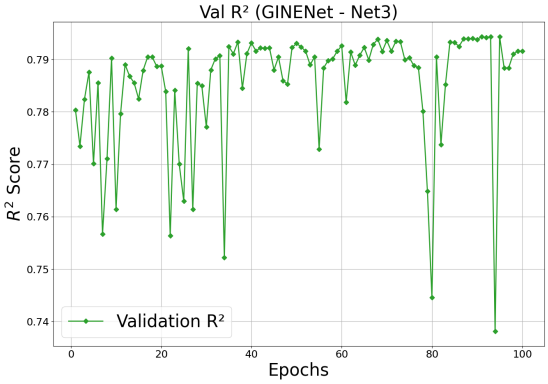
(a) Conv1D Loss Curves - Net3



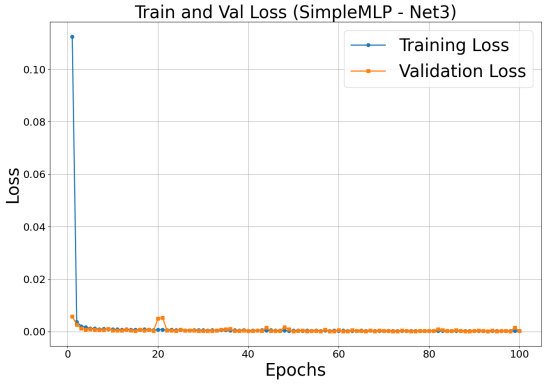
(b) Conv1D R^2 Scores - Net3



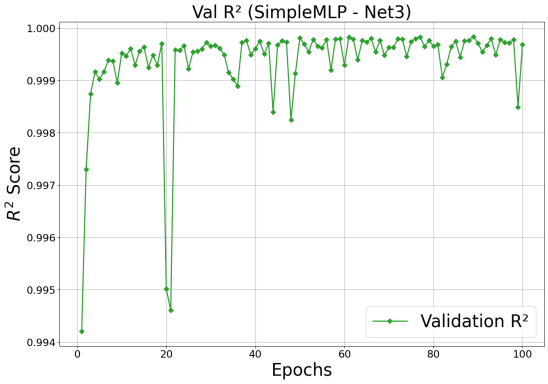
(c) GINENet Loss Curves - Net3



(d) GINENet R^2 Scores - Net3

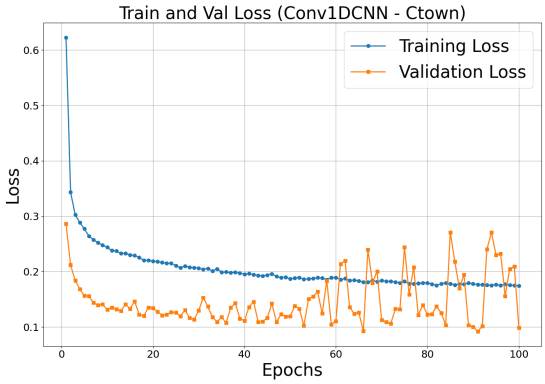


(e) MLP Loss Curves - Net3

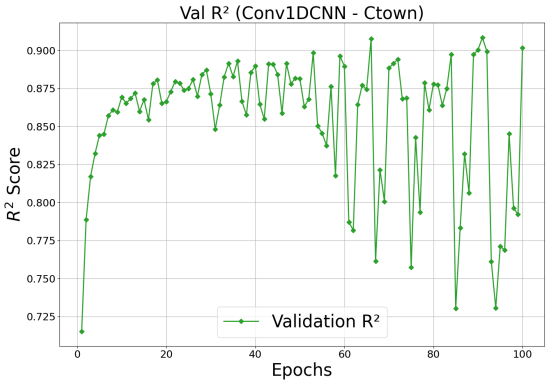


(f) MLP R^2 Scores - Net3

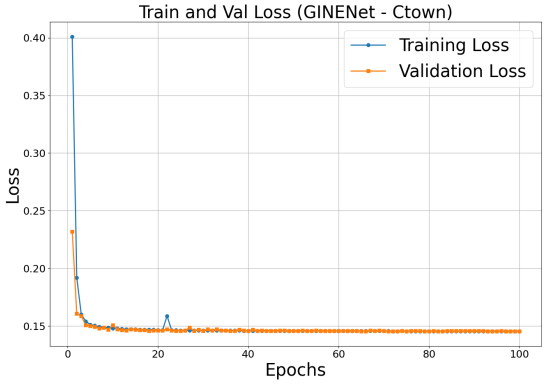
Figure 2.13: Training Loss Curves and R^2 Scores for Conv1D, GINENet, and MLP on the Net3 dataset.



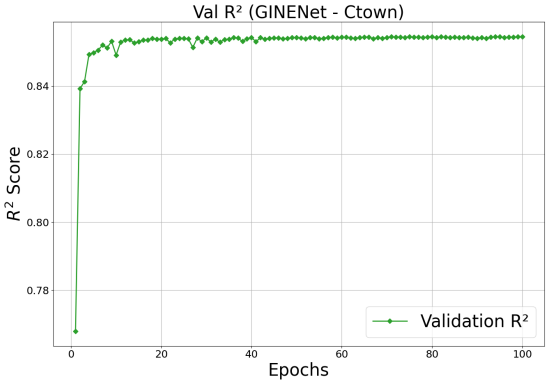
(a) Conv1D Loss Curves - C-Town



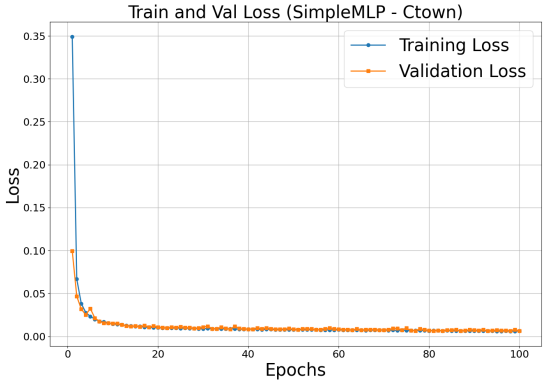
(b) Conv1D R^2 Scores- C-Town



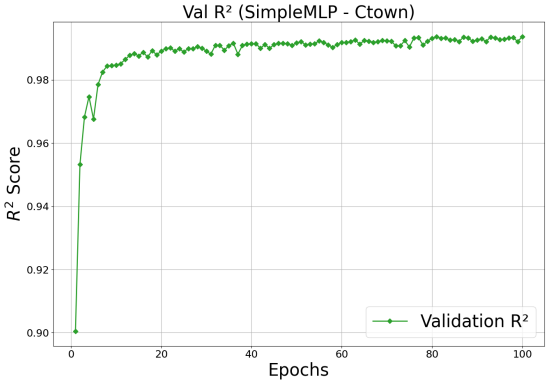
(c) GINENet Loss Curves- C-Town



(d) GINENet R^2 Scores- C-Town



(e) MLP Loss Curves- C-Town



(f) MLP R^2 Scores- C-Town

Figure 2.14: Training Loss Curves and R^2 Scores for Conv1D, GINENet, and MLP on the C-Town dataset.

Table 2.6: Comparison of test performance for SimpleMLP, Conv1DCNN, and GINENet on different WDN test sets. The metrics identified by "**" are calculated on unnormalized data, while the test loss remains scaled.

WDN	NN	Test Loss	Test MSE**	Test MAE**	Test R^2**
Hanoi	SimpleMLP	0.0001	0.0003	0.0133	0.9999
	Conv1DCNN	0.0019	0.0064	0.0603	0.9980
	GINENet	0.5055	3.2712	1.3719	0.4998
Net3	SimpleMLP	0.0001	0.0000	0.0024	0.9999
	Conv1DCNN	0.0024	0.0002	0.0107	0.9976
	GINENet	0.2056	29.4071	1.3352	0.7944
Ctown	SimpleMLP	0.0062	0.0233	0.0488	0.9874
	Conv1DCNN	0.0922	0.2497	0.1844	0.8662
	GINENet	0.1456	43.5870	2.3886	0.8545

For the Net3 dataset, the Conv1DCNN followed closely behind the MLP, with a test R^2 of 0.9976 and relatively low test MSE. However, on the larger, more complex C-Town network, the Conv1DCNN's performance decreased, with a test R^2 of 0.8662, indicating that while it remains competitive, it struggles to maintain high accuracy in more complex network structures.

The GNN, in contrast, performed significantly worse on the smaller datasets, particularly in the Hanoi network, where it achieved a test R^2 of 0.4998. However, in the larger and more complex C-Town network, the GNN showed better performance, with a test R^2 of 0.8545. This improvement suggests that the GNN's architecture can better leverage structural information in more complex networks, though it still underperforms compared to the MLP and Conv1DCNN.

These results underscore the importance of selecting the appropriate model based on the size and complexity of the network. Simpler architectures, such as MLPs, are highly effective for smaller networks, while graph-based models like GNNs provide advantages in larger, more complex networks where structural information is more critical. Additionally, these findings emphasize the potential benefits of exploring the optimal structure of these models. Techniques such as Grid Search [173] or similar methods could be invaluable in fine-tuning the models to better align with specific data types and features, ultimately improving performance.

Test Results with Leakages

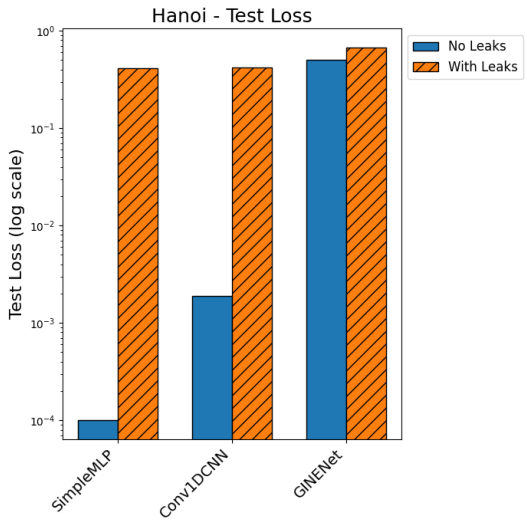
We evaluated the performance of the previously trained networks, which were trained on regular data without leakages, by testing them on unseen data containing newly introduced leakages. Using the best saved weights from earlier, we tested the models on Water Distribution Networks (WDNs) with simulated leaks. For the Hanoi and Net3 networks, leaks were added to 40% of randomly selected nodes using the native functions offered from the WNTR simulator, with each leak set to an area of 0.1 cubic meters, representing a continuous water leak across 1000 snapshots.

Table 2.7: Comparison of test performance for SimpleMLP, Conv1DCNN, and GINENet on different WDN test sets with added leakages. The metrics identified by "**" are calculated on unnormalized data, while the test loss remains scaled.

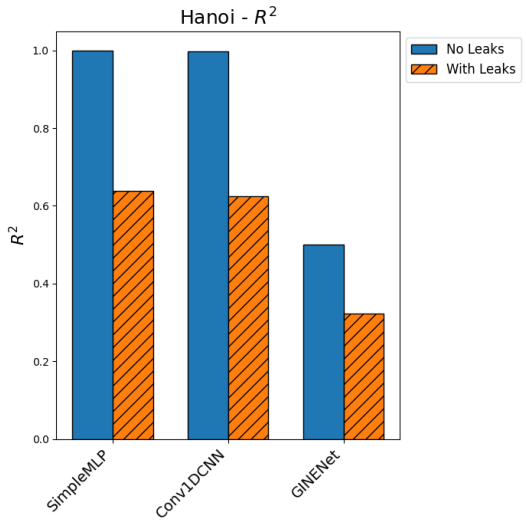
WDN	NN	Test Loss	Test MSE**	Test MAE**	Test R^2**
Hanoi (w/leaks)	SimpleMLP	0.4134	0.1276	0.2495	0.6393
	Conv1DCNN	0.4222	0.1328	0.2546	0.6246
	GINENet	0.6768	99.8288	6.2926	0.3230
Net3 (w/leaks)	SimpleMLP	1.2850	0.0003	0.0109	-0.2365
	Conv1DCNN	1.3010	0.0003	0.0111	-0.2646
	GINENet	0.5242	88.3696	6.2024	0.4758
C-Town (w/leaks)	SimpleMLP	0.2125	0.1408	0.2355	0.8683
	Conv1DCNN	0.3043	0.2687	0.2978	0.7487
	GINENet	0.9139	325.2015	15.3588	0.0865

In the case of C-Town, due to computational constraints, we introduced leakages in 10% of the nodes over 100 snapshots, with a reduced leak area of 0.001 cubic meters. In all cases, the leakages began at the start of the simulation and persisted throughout the simulation period.

Table 2.7 shows how each deep neural network (DNN) model exhibited a decrease in precision across all metrics when confronted with these unseen leakages, highlighting their lack of robustness in handling these kind of unexpected conditions.

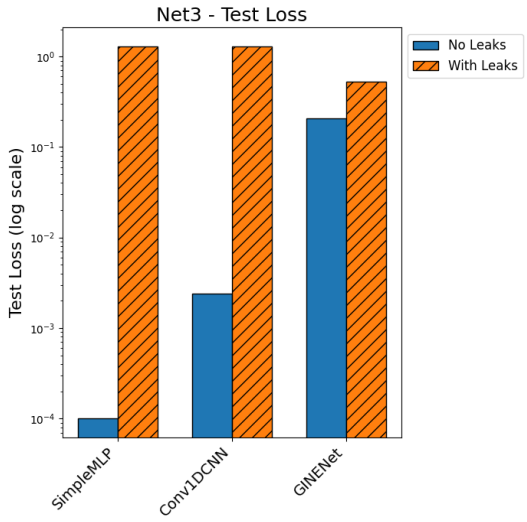


(a) Test Loss comparison (No Leaks vs. With Leaks) for Hanoi WDN. Higher test loss indicates higher prediction error.

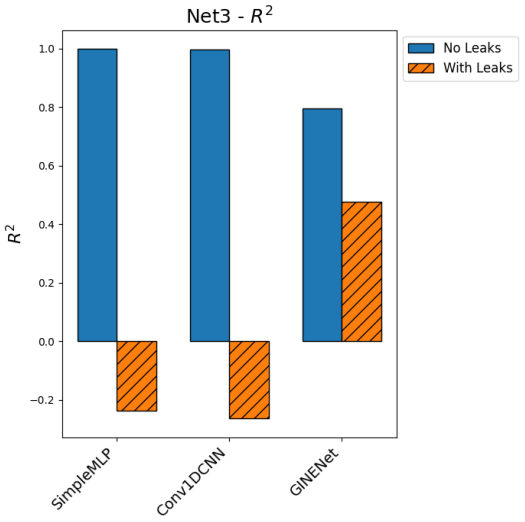


(b) R^2 comparison (No Leaks vs. With Leaks) for Hanoi WDN. Lower R^2 indicates less precise predictions.

Figure 2.15: Comparison of Test Loss and R^2 metrics for the Hanoi WDN under conditions with and without leaks.

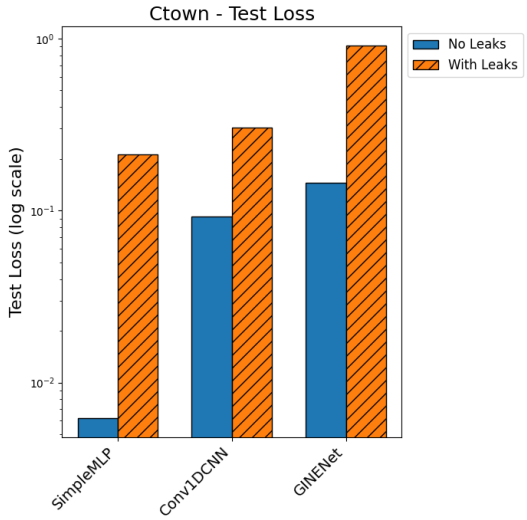


(a) Test Loss comparison (No Leaks vs. With Leaks) for Net3 WDN. Higher test loss indicates higher prediction error.

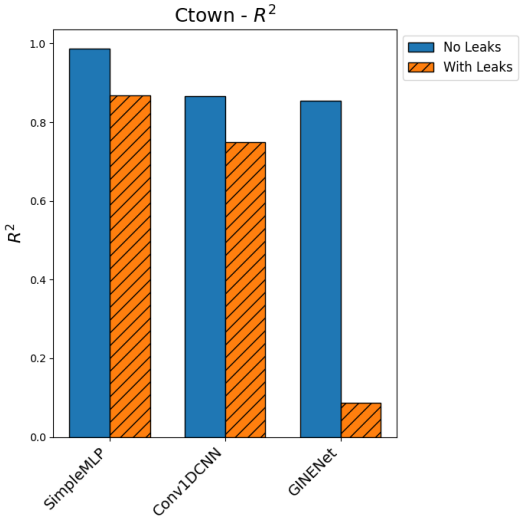


(b) R^2 comparison (No Leaks vs. With Leaks) for Net3 WDN. Lower R^2 indicates less precise predictions.

Figure 2.16: Comparison of Test Loss and R^2 metrics for the Net3 WDN under conditions with and without leaks.



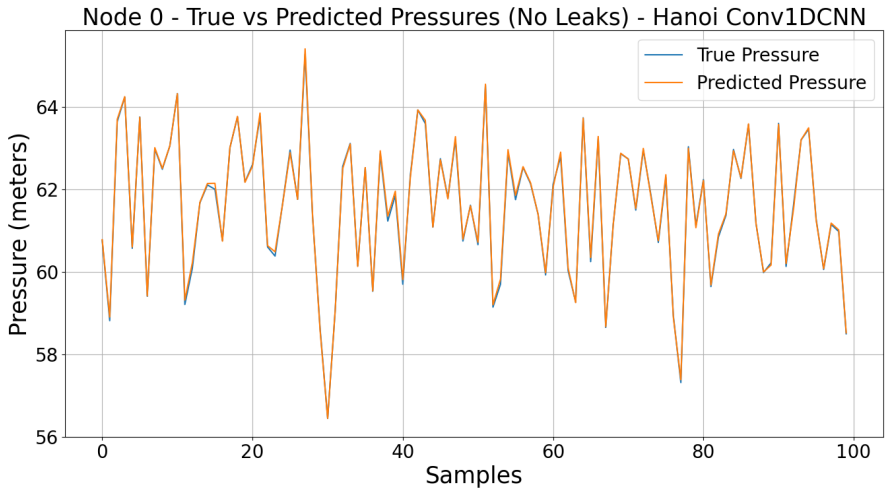
(a) Test Loss comparison (No Leaks vs. With Leaks) for Ctown WDN. Higher test loss indicates higher prediction error.



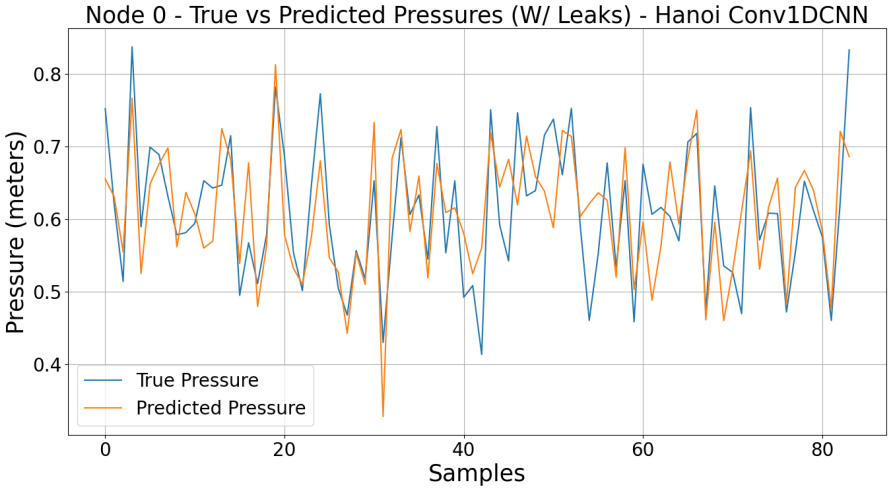
(b) R^2 comparison (No Leaks vs. With Leaks) for Ctown WDN. Lower R^2 indicates less precise predictions.

Figure 2.17: Comparison of Test Loss and R^2 metrics for the Ctown WDN under conditions with and without leaks.

As shown in Figures 2.15, 2.16, and 2.17, the test loss is consistently higher in the presence of leaks for all models across the three WDNs, indicating a greater prediction error. Conversely, the R^2 values are lower in the "With Leaks" condition, showing that the model's precision decreases when leaks are present. This trend is especially noticeable in the more complex models like GINENet, where the introduction of leaks significantly impacts both test loss and R^2.

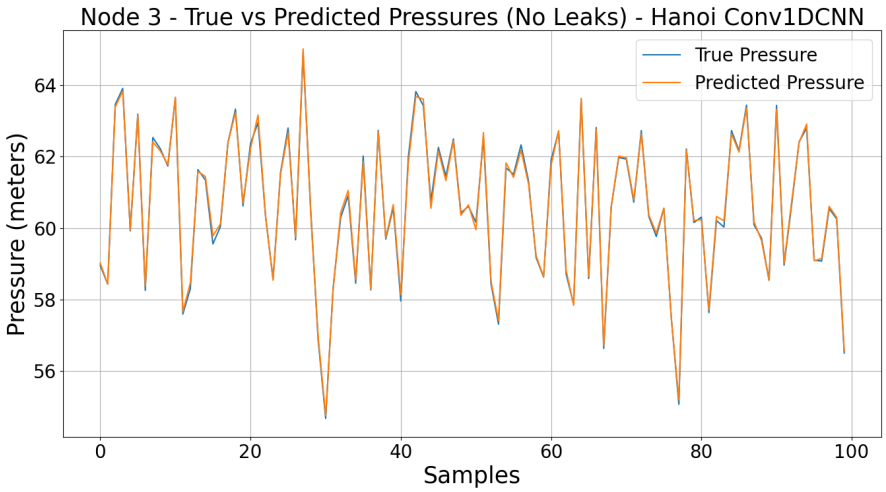


(a) Node 0 without leaks (Conv1DCNN on Hanoi WDN)

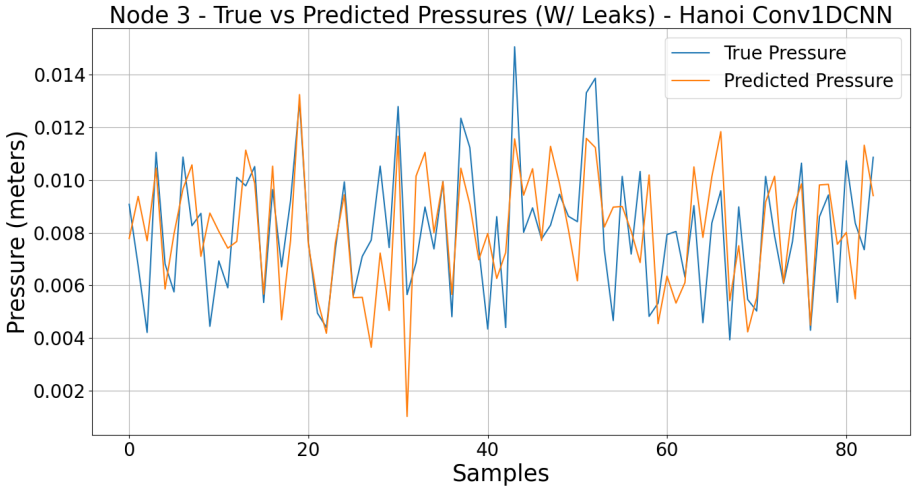


(b) Node 0 with leaks (Conv1DCNN on Hanoi WDN)

Figure 2.18: Comparison of predictions for Node 0 with and without leaks (Conv1DCNN on Hanoi WDN)

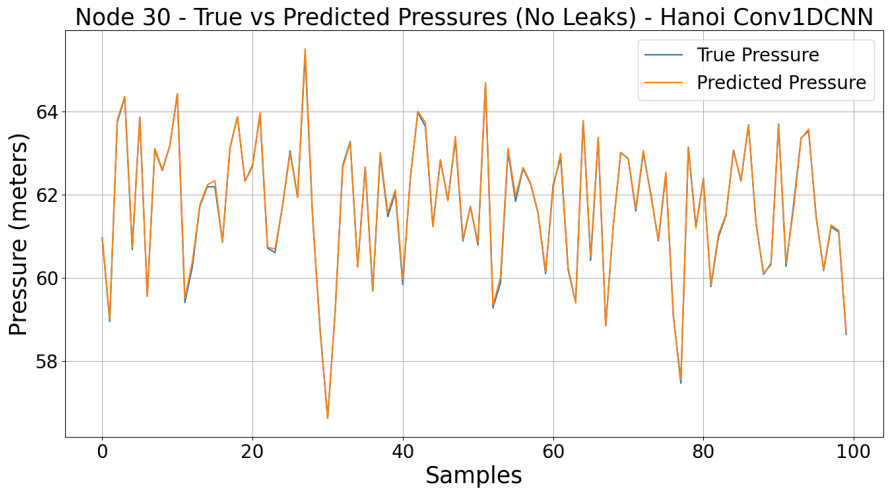


(a) Node 3 without leaks (Conv1DCNN on Hanoi WDN)

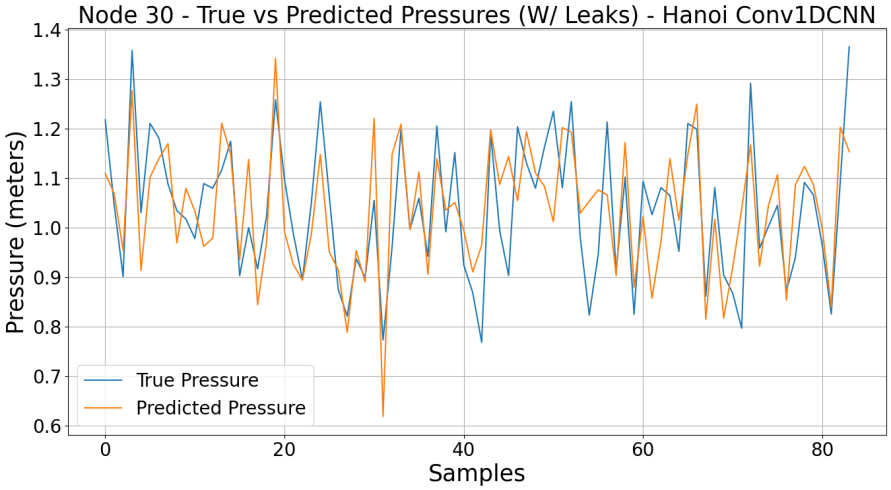


(b) Node 3 with leaks (Conv1DCNN on Hanoi WDN)

Figure 2.19: Comparison of predictions for Node 3 with and without leaks (Conv1DCNN on Hanoi WDN)



(a) Node 30 without leaks (Conv1DCNN on Hanoi WDN)



(b) Node 30 with leaks (Conv1DCNN on Hanoi WDN)

Figure 2.20: Comparison of predictions for Node 30 with and without leaks (Conv1DCNN on Hanoi WDN)

Figures 2.18, 2.19, and 2.20 display a comparison between the predictions made by the Conv1DCNN model and the true values for nodes with ID 0, 3, and 30 in the Hanoi WDN. These figures highlight two scenarios: predictions with no leakages and another where leaks are present.

When leakages are introduced, nodes 0 and 3 exhibit water losses, while node 30 remains unaffected. In the WNTR simulation, this lost water is represented as `leak_demand`. It's important to note that `leak_demand` is not included in the regular demand measured by the junctions, meaning the demand metric itself does not reflect the leak.

From these figures, it is evident that the presence of leaks significantly affects the model's predictions on pressures, even for non-leaking nodes like node 30. The predictions deviate substantially from the true values, underscoring the disruptive nature of leakage events on the model's performance.

These results indicate the need for further investigation into the impact of leakages on

WDNs and will guide future work in enhancing leak prediction, classification, and overall data analysis for water distribution networks.

2.6 Summary

This chapter explores the integration of IoT, edge computing, and ML within SWDNs, highlighting their significant potential for optimizing system performance and enhancing resilience.

We propose that hydraulic analysis can be offloaded to the edge for localized and efficient computation. We also discuss the challenges and solutions associated with edge processing, providing a comprehensive overview of its implementation.

The importance of effective GW placement in LoRaWAN networks is emphasized, as optimal positioning directly impacts data quality and overall system performance. Simulations conducted in environments like Hanoi, C-Town, and Net3 demonstrate how varying network characteristics affect data extraction rates, offering valuable insights for practical deployment strategies.

We evaluate several ML models, including MLPs, CNNs, and GNNs, for predicting pressure values from minimal input features. Each model performs differently based on the complexity of the WDNs, yet all show strong performance and generalizability. However, the introduction of random leakages exposes vulnerabilities, underscoring the need for more robust leak detection approaches.

Future work could focus on improving the robustness of leak detection models by refining the integration of NNs with real-time data streams. Additionally, exploring advanced graph-based models and hybrid cloud-edge architectures presents opportunities for enhancing system scalability and reliability.

Chapter 3

A Digital Twin Framework for Smart Water Distribution Networks Using Data-Driven Models

3.1 Introduction

In this chapter, we present an integrated framework that unites the theories and work discussed in previous sections, focusing on the intersection of Internet of Things (IoT) applications and Machine Learning (ML) analysis to address key challenges in a security-conscious manner while advancing critical system infrastructures. Water Distribution Networks (WDNs) are large-scale, complex, dynamic, and non-linear systems that must be effectively managed from design to execution to meet their objectives [174, 175, 176]. However, modern WDNs are increasingly difficult to manage due to urbanization, fluctuating consumer demands, and limited resources. Water utility managers face critical decisions in addressing challenges across various phases and time frames of WDN operations [177].

Our central contribution is the further transformation of WDNs into intelligent systems capable of real-time data analysis both at the edge and in the cloud, enabling fast intervention, management, ease of use, and rapid anomaly detection.

To solve these challenges we leverage the concept of Digital Twins (DTWs)—virtual replicas of physical systems—allowing for real-time monitoring and simulation. DTWs provide an innovative solution to modernize and simplify the management of water systems by transforming outdated infrastructures into enhanced digital replicas. This approach not only optimizes system performance but also prepares the infrastructure for future challenges such as sustainability and data-driven operations. **The goal is not to replace the physical system utilities but to enhance their capabilities and ease of maintenance. By integrating advanced frameworks for real-time data analysis, anomaly detection, ML, and IoT, our DTWs solutions empower both novice and expert users to manage complex systems effortlessly and make informed, rapid decisions.** From a networking perspective, we employ well-established and already discussed technologies like Long Range Wide Area

Network (LoRaWAN) to ensure reliable, scalable communication across these interconnected systems. This results in the creation of a comprehensive application, SWIM (Smart Water Interaction & Monitoring), which enables users of varying technical expertise to interact with IoT devices deployed in the context of a Smart Water Distribution Network (SWDN), process data streams efficiently, and maintain secure communication across the network.

All these challenges are key fundamental concepts of Industry 5.0 [57].

3.2 Digital Twin

A DTW is defined by three interconnected components: a real-world physical object, its digital counterpart, and the data flow that continuously links the two [178]. This concept has gained widespread application across various industries, including manufacturing, healthcare, aviation, smart cities, 6G networks, and transportation [179].

The understanding of DTWs can be categorized into three key models:

- **Static Mirror Model**: In this model, the virtual twin is a fixed representation of the physical object, created once and not updated with real-time data. While useful for basic visualizations, it lacks dynamic data exchange and does not adapt to changes in the physical object.
- **Unidirectional Simulation Model**: This model focuses on simulating the physical object digitally, with data flowing one-way from the physical object to the DTW. The virtual model is used to analyze and predict behavior, but there is no feedback loop influencing the physical object.
- **Integrated Feedback System**: The most advanced model, where the DTW functions as an adaptive, real-time system. Data flows bidirectionally, allowing the virtual twin to evolve based on real-time information from the physical object, and vice versa. This dynamic interaction enables continuous optimization and control.

A fully realized DTW operates as an intelligent, evolving system that monitors, manages, and optimizes the physical object throughout its lifecycle. The bidirectional data exchange between the physical and digital components allows for continuous feedback, making real-time adjustments possible. Physical objects can range from machines and infrastructure to entire smart cities.

Building on this concept, the Digital Twin Network (DTWN) expands the DTW into a network of interconnected digital twins. These interconnected systems facilitate real-time collaboration between multiple physical and virtual entities. Using advanced communication technologies, the DTWN ensures synchronized data flows and dynamic interactions, where both physical objects and their digital twins evolve and adapt collectively. This creates a collaborative network where multiple DTWs share data and complete complex tasks more efficiently.

A key distinction between traditional simulations and DTWs lies in their scope and data integration [180]. Simulations are typically limited to a single process and lack real-time feedback, while DTWs support continuous, two-way data flow. Real-time data from sensors is transmitted to the DTW, which processes this information to generate insights or actions that are fed back into the physical system. This ongoing exchange allows DTWs to simulate multiple scenarios in real-time, making them far more effective than traditional simulations in improving performance and solving complex problems across entire systems.

3.2.1 DT Communications

Effective communication is crucial for enabling DTWs and DTWNs, as it supports the continuous exchange of information across the network.

- **Physical-to-Virtual (P2V) Communications**: P2V communications enable the real-time transmission of data from a physical object to its virtual twin using wireless technologies such as LoRa and 5G/6G networks [181]. This connection allows the virtual twin to monitor, simulate, and provide feedback to the physical counterpart, ensuring dynamic interaction and synchronization.

- **Physical-to-Physical (P2P) Communications**: P2P communications facilitate data exchange between physical objects, employing devices like sensors, Radio Frequency Identifications (RFIDs), and controllers. These devices communicate via IoT gateways, Wi-Fi access points, and base stations using protocols such as Wireless Personal Area Networks (WPANs), ZigBee, and Low Power Wide Area Networks (LPWANs) technologies (e.g., LoRa and NB-IoT) [182]. This allows physical systems to interact and collaborate within the network.

- **Virtual-to-Virtual (V2V) Communications**: V2V communications occur in the digital domain, simulating interactions between the digital twins of physical objects. For example, in the Internet of Vehicles (IoV), V2V refers to the exchange of data between the digital twins of vehicles. Unlike P2P communications, V2V relies on the computational power of DTWs servers rather than wireless spectrum resources.

3.3 Background on DTW for WDNs

DTWs offer utilities precise and reliable data for analyzing the lifecycle of water distribution systems, assessing resilience through scenario testing, and evaluating asset health for proactive maintenance [183, 184]. In WDNs, DTWs are applied to address a broad spectrum of management challenges, from system design to operational decision-making. A case study presented by [180] demonstrated the significant impact of DTW implementation on reducing water losses, achieving a reduction of approximately 80% in real water losses. This led

to enhanced overall system efficiency, generating substantial economic benefits, with estimated savings of around EUR 165k and reduced the required annual water volume by 40% compared to previous operations, all while maintaining pressures above regulatory standards.

The primary capabilities of DTWs in WDNs include [179, 180]:

- Optimizing network element design to minimize environmental impact, particularly by reducing the carbon footprint, while ensuring compliance with quality-of-service standards.
- Managing physical assets through optimal renewal strategies, aimed at extending the lifecycle and reliability of the network's components.
- Implementing model-based leak detection and pre-localization techniques to minimize leak duration and reduce operational costs.
- Determining optimal daily operational parameters, such as water velocity, pressure, and energy efficiency, to maximize system performance.
- Providing early warning systems and facilitating informed, rapid responses to emergencies, thus supporting more effective decision-making.
- Simulating network behavior during both routine operations and maintenance activities to predict outcomes and improve operational planning.
- Simulating water demand and actual consumption using real-time sensor data on water levels, pressures, and flow rates, allowing for accurate network calibration and control.
- Ensuring the reliability of sensor data for real-time decision-making and the detection of critical alarms.

The development of a DTW for WDNs relies on continuous learning and adaptation, powered by large volumes of real-time field data stored on big-data platforms [184, 185].

The architecture proposed by [186] presents a DTW-based framework that integrates QGIS software plugins to improve decision-making in WDNs. This approach enhances logical reasoning, scalability, consistency, integrability, and efficiency in managing water distribution networks. However, their work lacks a detailed focus on the precision and effectiveness of ML models, and does not consider the use of established hydraulic simulation tools like EPANET or WNTR. Similarly, [187] propose leveraging Graph Convolutional Neural Networks (GCNNs) in conjunction with hydraulic models to develop DTWs for water networks. While their approach is promising, they do not provide a comprehensive tool for simple and effective WDN management. In contrast, [188] explore the use of deep neural networks (DNNs) as probabilistic surrogate models to replace full hydraulic equations and for leak detection. Although effective, this work also lacks a broader toolset for comprehensive WDN management.

3.4 A Comprehensive Application for IoT, Digital Twins, and Machine Learning: SWIM

In response to the increasing need for digital tools capable of managing complex, high-volume data in modern water systems, we introduce **SWIM (Smart Water Interaction & Monitoring)**, a DTW-based application. SWIM enables users to interact with a digital replica of a real-world WDN, offering real-time monitoring and analysis. Through an IoT LoRaWAN-based sensor network, each node in the WDN is equipped with authenticated IoT devices that stream hydraulic data at regular intervals directly to the platform.

Figure 3.1: SWIM logo.

SWIM integrates predictions from ML models, such as those discussed in Chapter 2, where results demonstrate their effectiveness in learning and predicting from extreme use cases and configurations. The platform allows users to compare real-time sensor data with these predictions through an intuitive interface that requires minimal technical expertise. Additionally, SWIM supports comparisons between ML predictions, real-time sensor data, and classical hydraulic simulation results from tools like Water Network Tool for Resilience (WNTR) and EPANET. This combination of capabilities offers a robust, innovative solution for managing the complexities of smart water distribution networks.

SWIM is currently under development. The following sections outline our proposed approach and present an overview of the key components of the platform. To evaluate the framework, we conduct a demonstrative representation within the previously mentioned simulated WDN, ”C-Town.”

3.4.1 SWIM development

SWIM is designed as a web application that operates on a local network and can be accessed through standard web browsers. While future updates may introduce changes, the current User Interface (UI) is modern, visually appealing, and fully responsive. A temporary (AI-generated) logo is shown in Figure 3.1.

Technology Stack: Flask, Python, HTML, JS, and CSS

SWIM’s back-end is developed in Python using the Flask framework [189]. The front-end layout is constructed with HTML (Figure 3.2a) and styled with CSS (Figure 3.2b), ensuring a clean and user-friendly interface. Core functionality for updating the UI and displaying real-time statistics is powered by JavaScript (Figure 3.2c). Specifically, the initialization, rendering, and representation of the WDN graph are custom-built using the D3.js library [190], allowing for dynamic and interactive visualizations.

Figure 3.2: HTML, CSS, and JS examples from the SWIM project.

```
{% extends "base.html" %}

{% block title %}SWDN analyzer{% endblock %}

{% block content %}
    <!-- Node Stats Popup -->
    <div id="node-popup" class="popup">
        <div class="popup-content">
            <span class="close" id="close-node-popup">&times;</span>
            <div class="node-title">Node Statistics</div>
            <!-- NODE STATS -->
            <div id="node-stats">
                <div class="static-info">
                    <div class="info-item">
                        <span class="info-label">Node ID:</span>
                        <span id="node-id" class="info-value"></span
                    </div>
                    <div class="info-item">
                        <span class="info-label">Coordinates:</span>
                        <span id="node-coordinates" class="info-valu
                    </div>
                    <div class="info-item">
                        <span class="info-label">Node Type:</span>
                        <span id="node-type" class="info-value"></sp
                    </div>
                </div>
                <table class="dynamic-info">
```

(a) HTML example from the SWIM project.

app > static > css > graph_content.css > ...

```
/* Main Content Styles */
#content-wrapper {
  display: flex;
  flex: 1;
  overflow: hidden;
  margin-top: var(--header-height);
  margin-left: 0;
  transition: margin-left 0.3s ease;
}

#resizable-container {
  display: flex;
  width: 100%;
  height: 100%;
  position: relative;
}

/* Graph Container Styles */
#graph-container {
  flex: 1;
  position: relative;
  overflow: hidden;
  background-color: white;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

(b) CSS example from the SWIM project.

app > static > js > graph.js > createGraph

```
155
156 function createGraph(graph) {
157     const container = document.getElementById('graph-container');
158     const { width, height } = updateSVGSize(container);
159
160     // Adjust NODE_SCALE to start smaller
161     const nodeCount = graph.nodes.length;
162     const avgNodeSize = Math.min(width, height) / Math.sqrt(nodeCou
163     NODE_SCALE = Math.max(0.3, Math.min(1.0, avgNodeSize / 90)); //
164
165     // Dynamically calculate LINK_STROKE_WIDTH based on the graph s
166     LINK_STROKE_WIDTH = Math.max(2, Math.min(10, avgNodeSize / 30))
167
168     simulation = d3.forceSimulation(graph.nodes)
169         .force('link', d3.forceLink(graph.links).id(d => d.id))
170         .force('charge', d3.forceManyBody())
171         .force('center', d3.forceCenter(width / 2, height / 2))
172         .stop();
173
174     graph.nodes.forEach(node => {
175         if (node.x != null && node.y != null) {
176             node.fx = node.x;
177             node.fy = node.y;
178         }
179     });
```

(c) JS example from the SWIM project.

The database for managing registered IoT devices is structured using SQL, which supports efficient data storage and retrieval.

The whole SWIM stack is represented in Figure 3.3.

Eclipse Ditto Integration and IoT Security

Eclipse Ditto [191] is integrated into SWIM to manage digital twins for IoT devices, abstracting physical devices into digital representations. This allows users to interact with and monitor these digital counterparts through a unified API, enabling real-time control and monitoring of IoT systems with both synchronous and asynchronous communication.

A key feature of Eclipse Ditto is its state management system, which maintains synchro-

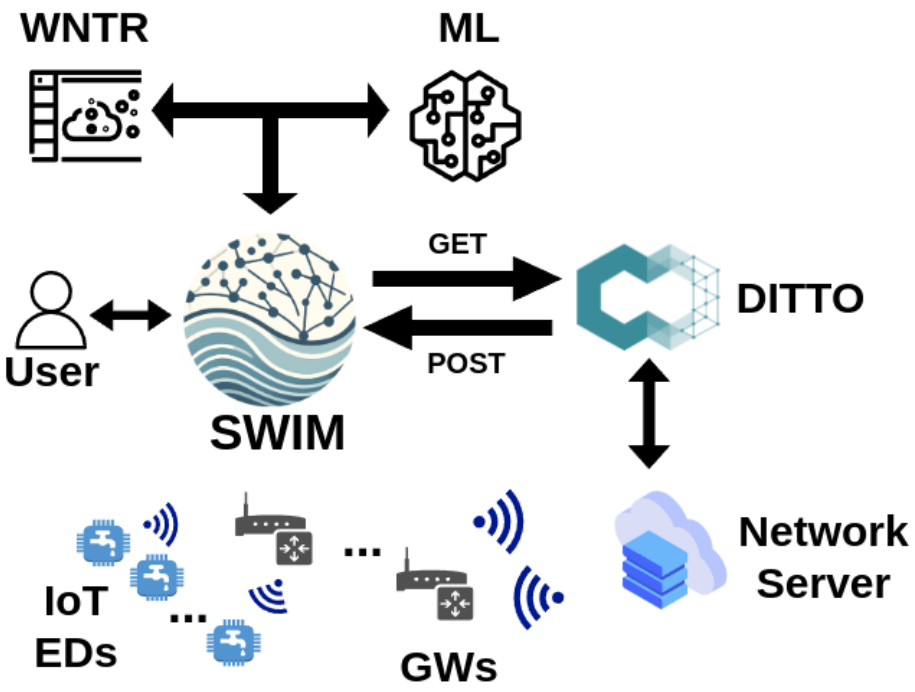


Figure 3.3: SWIM technology stack integrating IoT End Devices (EDs) and Gateways (GWs) with SWIM for data collection and analysis. The system uses ML models, WNTR, and interacts with DITTO and the network server via GET/POST requests.

nization between a device's desired, reported, and actual states. This ensures that digital twins reflect the real-world behavior of their physical counterparts accurately. Ditto's flexible, protocol-agnostic architecture supports MQTT, HTTP, and Kafka, making it adaptable to various IoT domains, from industrial systems to smart cities.

Eclipse Ditto also integrates robust security features:

- **Authentication and Authorization**: Ditto supports OAuth 2.0, JSON Web Tokens (JWT), and basic authentication. Detailed access control policies enable role-based access, and Ditto can integrate with external identity providers like Keycloak for secure OpenID Connect (OIDC) authentication.

- **Secure Communication**: To protect data in transit, Ditto supports Transport Layer Security (TLS) for its messaging protocols (AMQP, MQTT, HTTP), and client certificates for mutual TLS (mTLS), ensuring secure communication between clients and servers.

- **General Security Practices**: Eclipse Ditto follows industry-standard security practices, adhering to the Eclipse Foundation's vulnerability management policies. It employs the "principle of least privilege" for access control and supports immutable objects, enhancing security in distributed, event-driven environments.

These security features ensure that interactions with digital twins remain secure, while Ditto's adaptable design allows it to handle a variety of real-world IoT applications efficiently.

3.4.2 SWIM Functionalities

SWIM provides a range of interfaces to assist users in managing and analyzing their SWDNs. Although SWIM is still under development, this section outlines the key components and core concepts that form the foundation of the tool.

Main Interface

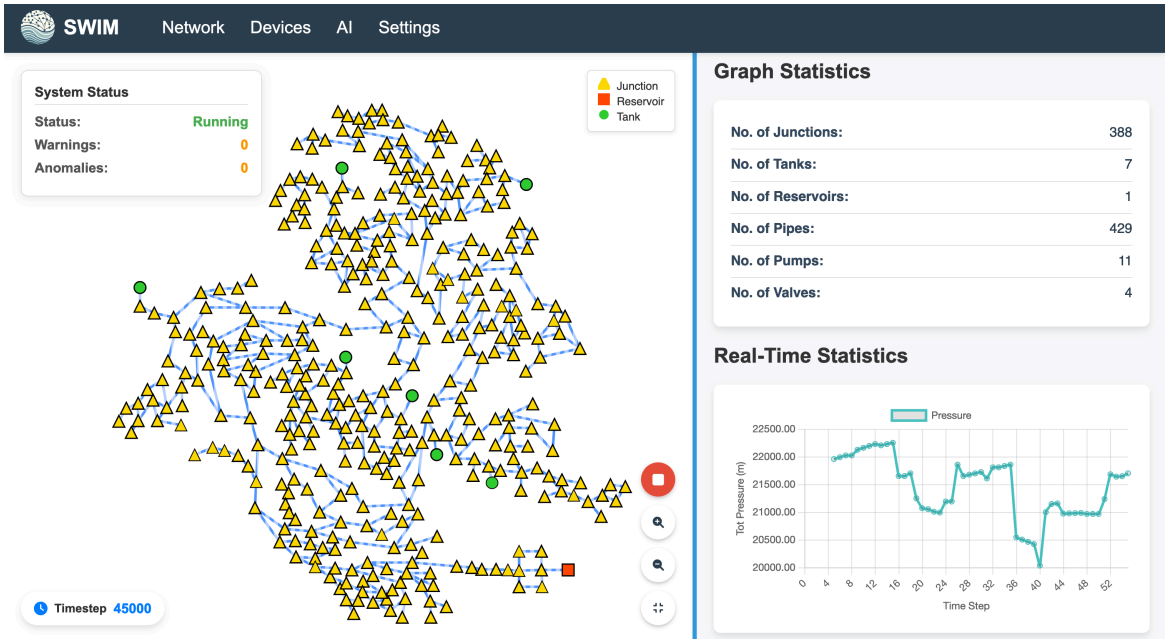


Figure 3.4: SWIM home page displaying the C-Town WDN.

The main interface of SWIM (Figure 3.4) presents a digital twin graph of the real WDN, allowing users to manage and interact with the system. For demonstration purposes, the C-Town network, discussed in previous sections, is used in this demo.

Users can interact with the WDN graph by clicking on individual nodes, resizing the graph, and adjusting the layout dynamically. A sidebar on the right provides real-time updates and displays both current and historical data about the WDN. This ensures that users have access to critical information for monitoring and decision-making, all within an intuitive and interactive interface.

WDN Node Analysis

Each node in the WDN graph is classified according to its type—Junctions, Reservoirs, or Tanks—following the EPANET/WNTR classification system. By clicking on any node, users can access an interactive popup menu that provides detailed analysis and management options for that specific node (Figure 3.5). This interface displays static information such as the Node ID, coordinates, and type, along with a table presenting hydraulic data from three different sources.

The ”SIM” column shows data derived from classical hydraulic simulations, performed using WNTR. These simulations are based on digital representations of the WDN, converted

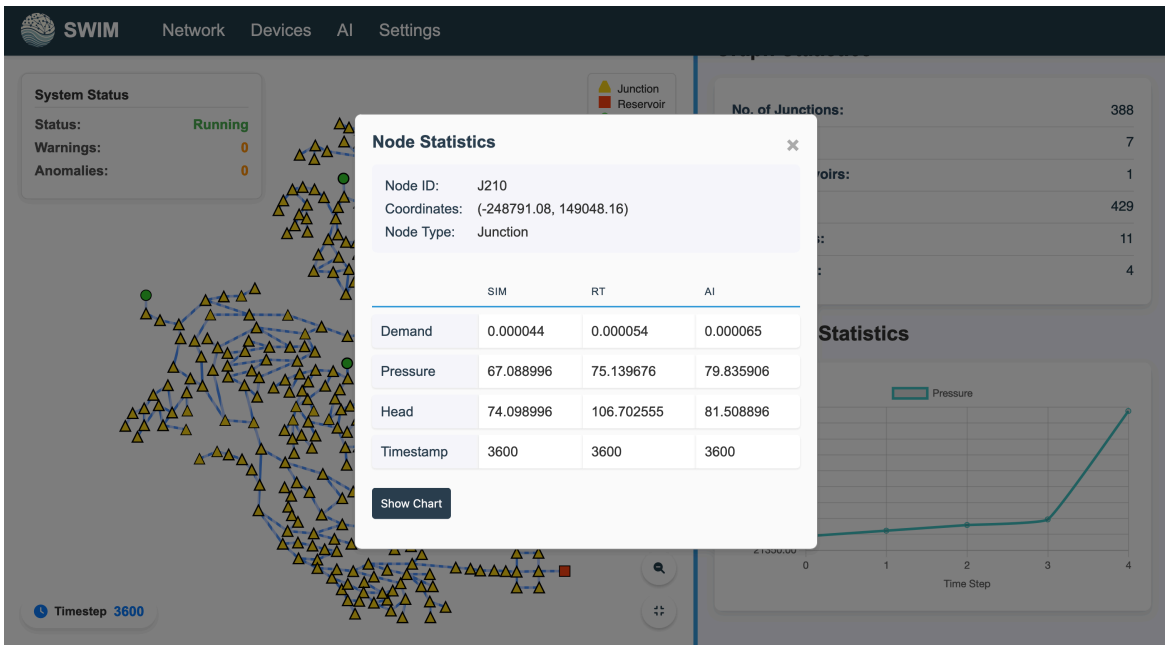


Figure 3.5: SWIM popup displaying the properties of node ”J373”. The table presents hydraulic data from three sources: the ”SIM” column shows values obtained from the WNTR simulator, the ”RT” column displays real-time data collected from IoT meters, and the ”AI” column provides predictions generated by the data-driven model for the corresponding snapshot.

into ”.inp” files, which serve as inputs for EPANET. The ”RT” column presents real-time data gathered from IoT devices registered to that node, with data exchanged via the Eclipse Ditto framework. Finally, the ”AI” column provides predictions from data-driven models seen on Section 2.5, estimating values based on initial conditions at the selected timestamp. This predictive capability is valuable for extended analyses, such as leak detection or system optimization.

IoT Device Management

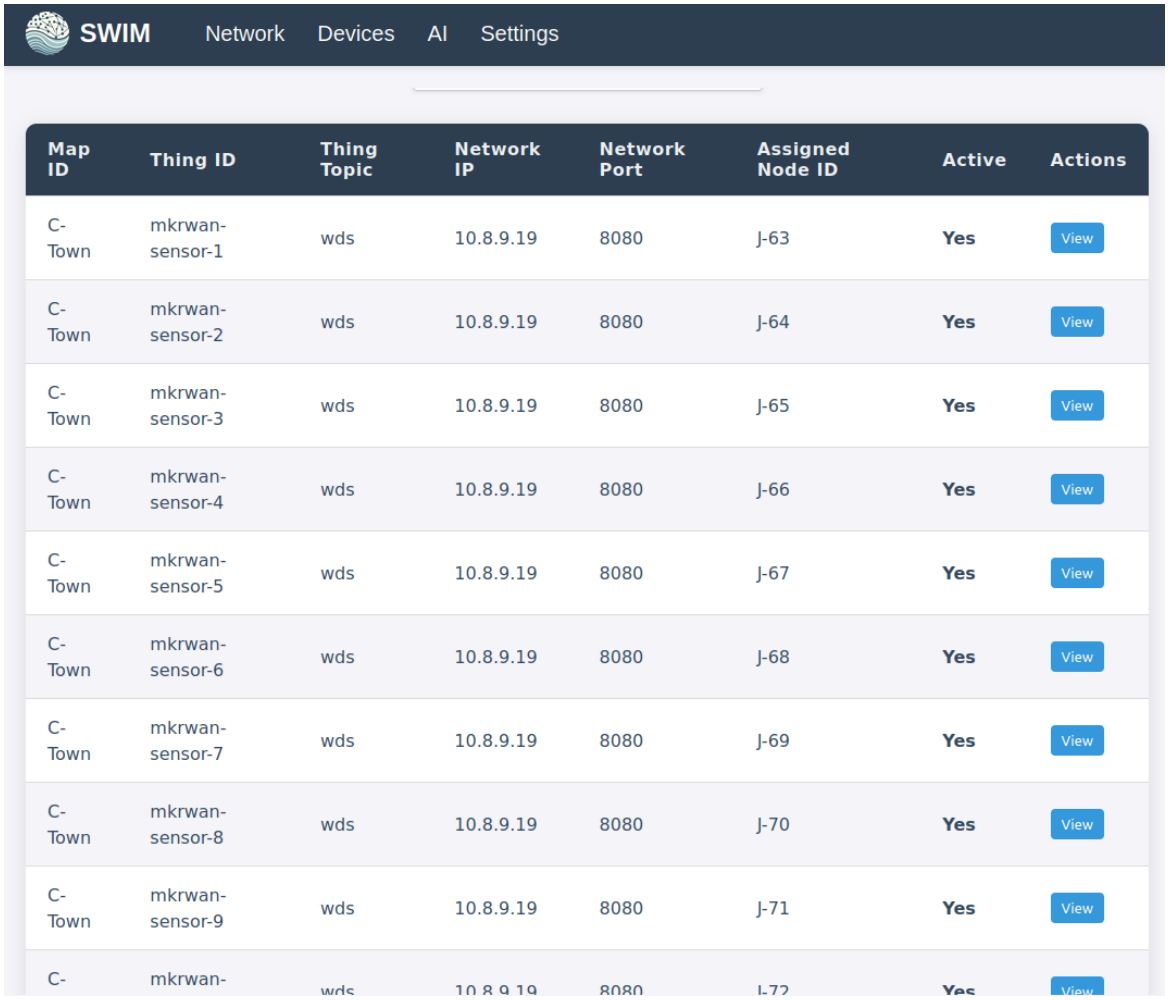


Figure 3.6: SWIM IoT device management page.

SWIM also offers a dedicated interface for managing IoT devices registered within the WDN (Figure 3.6). Device properties are displayed in a table, listing each device's ID, topic, network IP, network port, assigned node ID, and current status (active/inactive). Users can analyze or manage individual devices by interacting with the corresponding action buttons. All listed information is necessary for Eclipse Ditto functionalities, simplifying device management and interaction for users without advanced technical knowledge. This functionality allows users to send GET and POST requests seamlessly through the Ditto framework.

Future Steps: Data Analysis and Real-World Deployment

Future iterations of SWIM will include enhanced data visualization and analysis features (Figure 3.8). These functionalities will enable users to generate detailed plots and charts, supporting in-depth analysis of historical data. Users will be able to evaluate individual nodes or perform comprehensive assessments of the entire Water Distribution Network (WDN), gaining valuable insights into network performance. This capability will facilitate more informed decision-making based on trends and system behaviors observed over time.

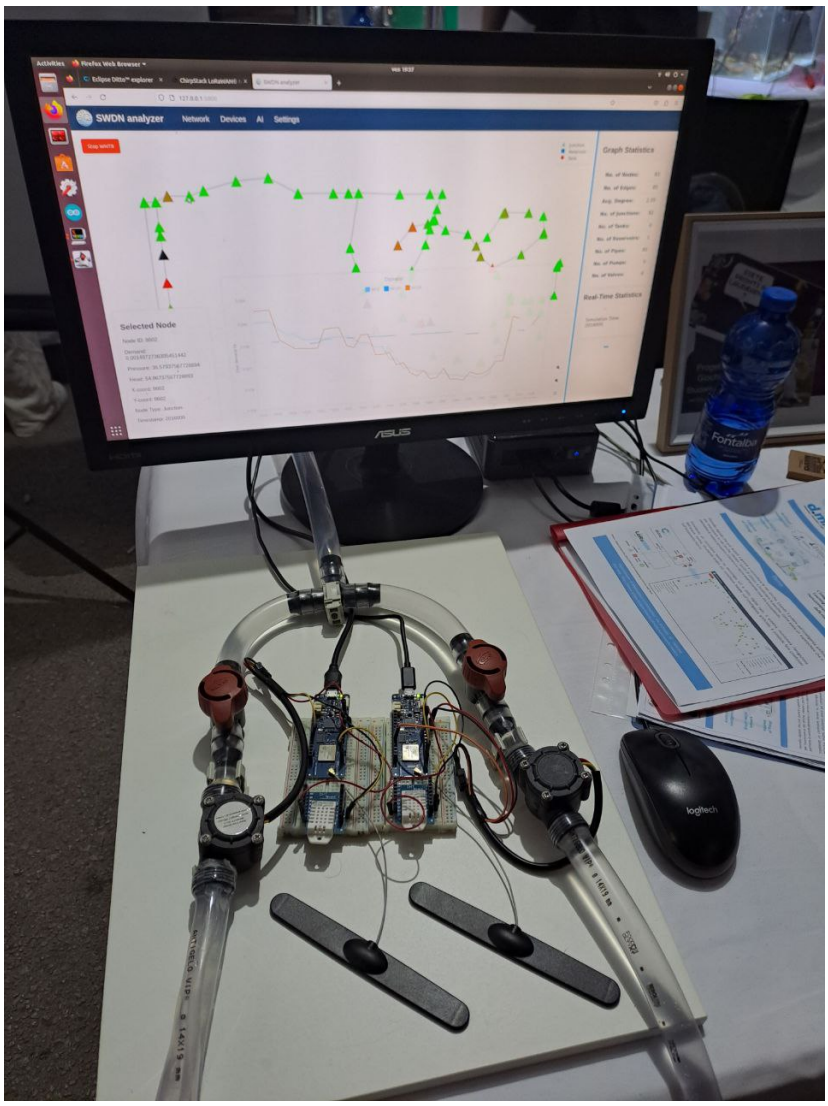


Figure 3.7: Real-world demo of SWIM.

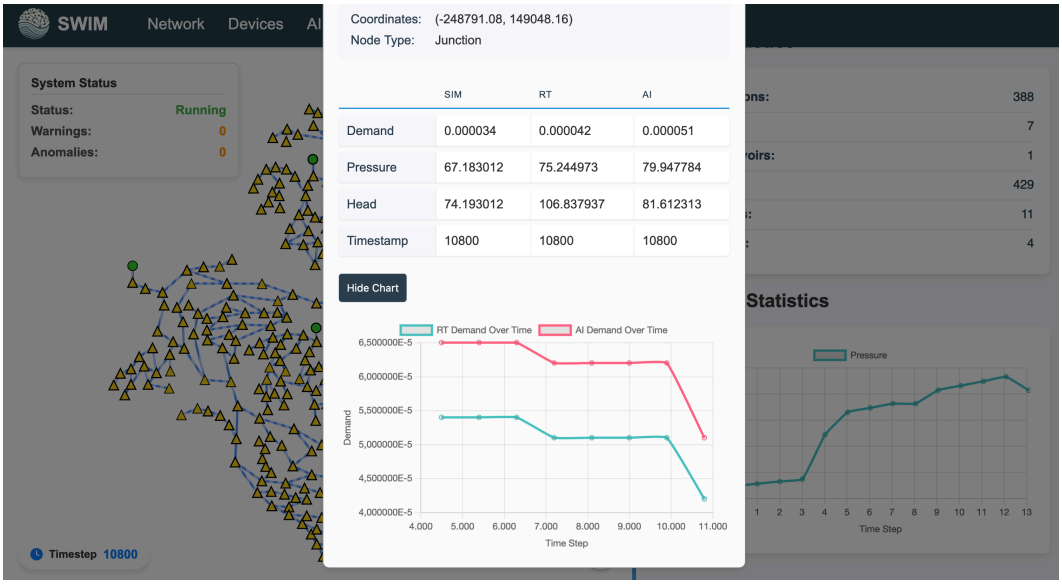


Figure 3.8: Preview of experimental SWIM functionalities for data visualization and analysis.

Additionally, we plan to extend SWIM's functionality to real-world deployments once the platform is fully developed. As a proof of concept, we have already demonstrated SWIM's integration with real devices, such as data collection from an Arduino through Ditto, and performed interactive predictions using the previously discussed DNN models. Figure 3.7 shows a snapshot of this real-world demo, marking a significant step toward a complete real-world implementation of SWIM.

3.5 Summary

This chapter demonstrates an integrated framework that leverages IoT, DTWs, and ML to optimize the overly complex management of WDNs. By implementing DTWs, we enable real-time monitoring, simulation, and data-driven decision-making, addressing the complex challenges of modern water distribution systems. The SWIM platform, developed as part of this research, provides an innovative tool for managing WDNs through IoT-enabled sensors, real-time data analysis, and ML-based predictions. This framework sets the foundation for further advancements in smart infrastructure management, ensuring scalability, efficiency, and security in critical water systems.

Chapter 4

Secure authentication for constrained IoT devices

4.1 Introduction

IoT devices, equipped with sensors, software, and connectivity components, facilitate communication with other devices and systems over the internet, making them integral to both industrial and everyday environments. This continuous connectivity heightens the need for secure communication and authentication, as IoT systems are particularly vulnerable to security risks and attacks due to their large-scale deployments, limited computational capabilities, and often inadequate security measures by users—such as neglecting to change default passwords. Consequently, numerous security solutions have been proposed in the literature to address these vulnerabilities [192, 193, 194, 195, 196].

Unlike traditional devices, IoT devices face unique security challenges, particularly due to their ability to change environments or ownership. For instance, when a smart home is sold, all embedded IoT devices must erase historical data and reset to allow control by the new owner. These lifecycle management and privacy concerns are significant, as IoT devices often handle sensitive personal data, which necessitates robust management practices beyond real-time data exchange [197]. Our work focuses specifically on authentication and key agreement protocols, crucial for maintaining security when devices move across domains and change ownership, ensuring that long-term secrets are not stored within the entities that manage these environments.

A key strategy in addressing these challenges is to leverage established solutions rather than invent new mechanisms from scratch. The concept of device mobility and ownership change has long been managed in cellular networks, starting from 2G/GSM and evolving through 3G, 4G, and now 5G systems. These networks employ an authentication and key agreement (AKA) framework that distinctly separates the roles of maintaining long-term secrets and verifying authenticity (”home” entity) from performing real-time authentication (”serving” entity), which operates using pre-distributed credentials without direct access to long-term secrets [198].

In this chapter, we introduce RIOT-AKA, an authentication and key agreement protocol inspired by the AKA mechanisms from cellular networks, tailored for IoT environments. This contribution includes the design, a proof-of-concept implementation on the RIOT secure IoT Operating System [199, 200], and an experimental evaluation of the protocol. RIOT-AKA is optimized for IoT by utilizing symmetric cryptography, specifically HMAC-SHA256, to minimize processing demands and resource usage, achieving mutual authentication with just two message exchanges. The protocol is designed to fit a deployment model where local entities, such as gateways, manage the authentication handshake, while the actual validation of devices—via long-term secret storage and the provision of temporary credentials—is handled by a separate stakeholder, such as an IoT service provider or the device manufacturer.

While RIOT-AKA shares conceptual similarities with cellular network AKA protocols, it introduces key technical distinctions. Unlike cellular systems that use specialized algorithms like MILENAGE and TUAK standardized by 3GPP [201], RIOT-AKA employs HMAC-SHA256, a readily available cryptographic function in RIOT. This approach simplifies the protocol, removes the necessity for backward compatibility, and eliminates the need to embed additional signaling information in authentication messages, making the design more straightforward and efficient.

It should be noted that RIOT-AKA is currently a preliminary proof-of-concept. Advanced features, such as sequence number anonymization and the generation of ephemeral encryption keys, are not yet implemented and are planned for future enhancements.

Additionally, as an exploratory aspect of this chapter, we investigate the use of Physical Unclonable Functions (PUFs) to manage long-term secrets. By utilizing the inherent randomness of SRAM cells in IoT devices running RIOT OS, we propose a method for generating and managing secrets that are never stored in the device's memory, thus enhancing security by ensuring that sensitive information remains transient and inaccessible.

The research and developments discussed in this chapter have been previously published [202].

4.2 Background

4.2.1 RIOT-OS

Figure 4.1: RIOT-OS logo.

The RIOT Operating System (OS) is a lightweight, open-source OS designed specifically for the needs of networked, memory-constrained IoT devices. It is built on a microkernel architecture that supports full multithreading and real-time capabilities, making it suitable for a wide range of IoT applications. RIOT OS is distributed under the GNU Lesser General Public License (LGPL), which allows for broad use and modi-

fication within the IoT development community [203, 204]. Although not fully compliant with the Portable Operating System Interface (POSIX) standard, RIOT OS provides robust application development support in C and C++, along with an extensive collection of commonly used libraries, such as WolfSSL. The OS features a comprehensive TCP/IP network stack that includes support for IPv6, 6LoWPAN, and various IoT protocols like RPL and CoAP, enabling seamless connectivity in IoT environments [204, 199]. Additionally, RIOT offers essential utilities, including cryptographic libraries, various data structures (e.g., bloom filters, hash tables, and priority queues), and a shell environment for enhanced functionality.

RIOT OS's microkernel architecture includes critical OS components such as thread management, a priority-based scheduler, APIs for inter-process communication (IPC), system timers, and mutexes, which together ensure efficient multitasking and resource management. The flexibility of RIOT OS extends to development on Linux and macOS environments, where applications can be built, debugged, and tested using tools like GCC, GDB, and Valgrind. Developers can utilize native emulation, allowing RIOT to run as a process on these operating systems, thereby streamlining the development and testing processes without the need for dedicated hardware [204].

RIOT OS stands out due to its modularity, support for real-time operations, and compatibility with a wide range of IoT protocols and tools, making it a versatile choice for developing IoT solutions that require reliable, low-power, and networked capabilities.

4.2.2 Cellular network architecture

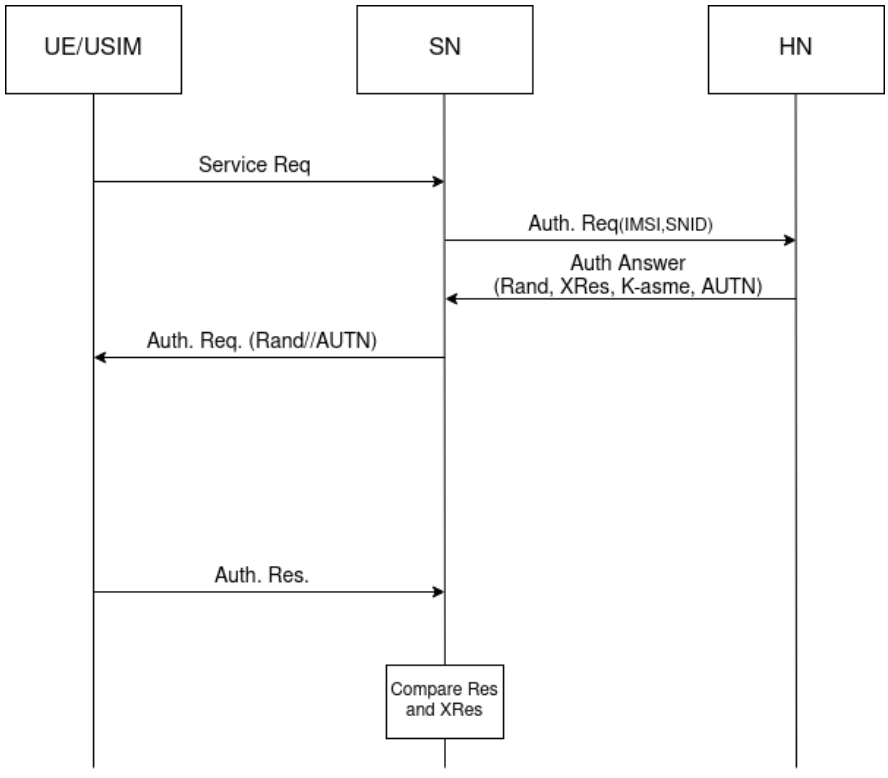


Figure 4.2: Basic 4G-AKA Authentication handshake.

The architecture of cellular networks [205] primarily revolves around three key entities: the User Equipment (UE), such as smartphones or IoT devices equipped with SIM cards; the Home Network (HN), which manages subscriber databases and authenticates users; and the Serving Network (SN), which UEs connect to and which facilitates communication with the HN. Each UE's SIM card is equipped with cryptographic capabilities, such as symmetric encryption and MAC generation, and stores crucial information including a unique subscriber identifier known as the International Mobile Subscriber Identity (IMSI), a unique symmetric key shared with the HN, and a 48-bit sequence counter designed to prevent replay attacks. Both the IMSI and the symmetric key are established during SIM initialization and remain constant throughout the SIM's lifecycle. Correspondingly, the HN maintains a database containing this information for each of its subscribers' SIM cards.

To establish a secure communication channel, a UE must authenticate with both its HN and SN, ensuring that user data remains confidential. This is achieved through the Authentication and Key Agreement (AKA) protocol [198], which facilitates a secure key exchange between the UE and its HN, mediated by the SN. The AKA protocol uses the shared secret key and sequence counter stored on the SIM and in the HN's database to perform a challenge-response mechanism among the three entities, thereby authenticating the UE and safeguarding the communication (see Fig. 4.2).

4.2.3 Constrained Application Protocol (CoAP)

The Constrained Application Protocol (CoAP) [16, 17] is a specialized web transfer protocol designed by the Internet Engineering Task Force (IETF) for constrained nodes and networks, typically found in Internet of Things (IoT) environments. CoAP is based on the Representational State Transfer (REST) architecture, similar to HTTP, but optimized for constrained devices and low-bandwidth networks. It operates over the User Datagram Protocol (UDP), which allows it to maintain a lightweight and efficient communication model suited to the limitations of small-scale, power-constrained devices.

CoAP facilitates Machine-to-Machine (M2M) communication by enabling devices, known as nodes, to communicate both within constrained networks and with the broader internet. Unlike traditional HTTP, which operates over TCP, CoAP's use of UDP reduces overhead and supports multicast, which is advantageous in networks with limited resources. CoAP messages are encoded in a concise binary format, which further minimizes the communication footprint, making it suitable for environments where energy conservation and bandwidth efficiency are critical.

The protocol structure of CoAP includes two main layers: the messaging layer and the request/response layer. The messaging layer ensures reliability and handles the transmission of messages using four types of messages: Confirmable (CON), Non-Confirmable (NON), Acknowledgment (ACK), and Reset (RST). This layer is responsible for managing message delivery and retransmissions when necessary. The request/response layer manages the actual communication between clients and servers, supporting synchronous and asynchronous

interactions. It includes mechanisms such as Piggy-Backed responses for immediate replies, and Separate responses for delayed processing, which help in maintaining communication efficiency and reliability.

One of CoAP's key features is its ability to support asynchronous message exchanges, which is particularly useful in IoT scenarios where devices may not always be ready to respond immediately. Furthermore, CoAP integrates well with existing web technologies, allowing seamless interaction between constrained devices and web services, thereby enabling a broader range of applications in smart environments, home automation, and industrial settings.

CoAP also incorporates security features through Datagram Transport Layer Security (DTLS), which provides essential security mechanisms such as authentication, data integrity, and confidentiality. This security model ensures that CoAP can be used securely even in environments where data protection is crucial.

CoAP's lightweight design, efficient use of resources, and robust feature set make it an ideal protocol for IoT applications, particularly where devices operate under constraints of power, processing, and bandwidth.

4.3 The RIOT-AKA Authentication Framework

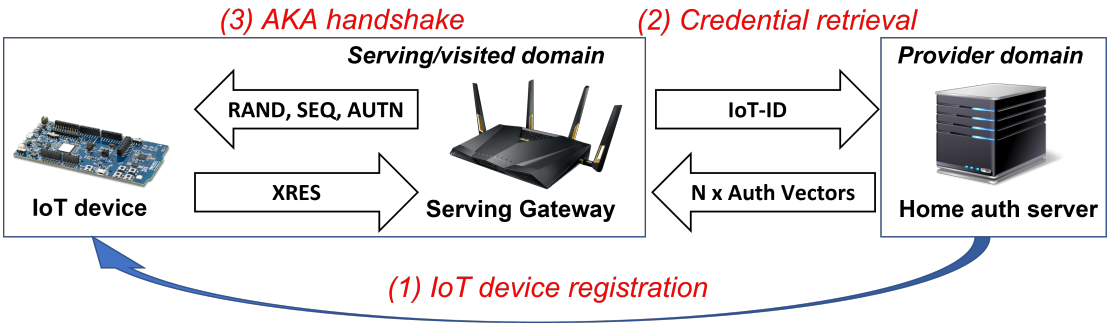


Figure 4.3: RIOT-AKA deployment framework.

The RIOT-AKA authentication framework, illustrated in Figure 4.3, operates with three main entities: the IoT device, the "serving" domain where the device is deployed, and the "provider" cloud domain. In this setup, a gateway within the serving domain functions as a local authentication server, utilizing credentials obtained offline from the provider's home authentication server.

The framework's operation is divided into three distinct phases:

1) IoT Device Registration: Initially, the IoT device is provisioned with a secret key K_0, which is typically generated and stored by the provider's home authentication server alongside the device's unique identifier. We also provide an alternative method that generates the secret key K_0 using an SRAM Physically Unclonable Function (SRAM-PUF) available through the RIOT OS.

2) Retrieval of Authentication Credentials by the Serving Gateway: This phase mirrors the credential retrieval process in cellular networks, though it differs in the specifics of how authentication credentials are computed. When an IoT device enters or is deployed in a serving domain, the serving gateway retrieves a set of N authentication vectors from the home authentication server. The parameter N determines the maximum number of authentication attempts that the device can perform before new credentials are required. This process is repeated whenever the existing set of credentials is exhausted. For testing purposes, our implementation uses a default value of $N = 100$, though the optimal setting for N depends on factors like device mobility, performance requirements, and security considerations, which are beyond the scope of this discussion.

3) Run-time Authentication and Key Agreement Handshake: This online phase occurs during the actual authentication or re-authentication of the IoT device. Similar to cellular networks, this handshake over the air interface achieves mutual authentication using only two messages, as shown in Figure 4.3. In the first message, the gateway sends a random challenge, the sequence number for the current authentication, and an authentication token (AUTN) that allows the IoT device to verify the gateway's authenticity, confirming that it has received valid credentials from the home authentication server. The IoT device then responds with a message (XRES) that serves as proof of its authenticity.

4.3.1 Protocol Design Details

Our RIOT-AKA protocol diverges from the traditional 4G-AKA protocol primarily in two aspects: the transport layer and the cryptographic algorithms employed. Firstly, our design utilizes the Constrained Application Protocol (COAP) defined in RFC 7252 [16], which is well-suited for IoT environments, as the transport layer for authentication messages. Secondly, due to the lack of support for 3GPP-specific algorithms like MILENAGE and TUAK in RIOT OS, we rely on cryptographic algorithms that are commonly supported within the IoT ecosystem, specifically HMAC-SHA256.

The core of our design uses HMAC-SHA256 for multiple purposes: (i) generating the authentication token and the corresponding device response, and (ii) deriving ephemeral master keys for encryption and integrity, which are utilized by the protocol-specific IoT/RIOT cipher suites. Although the same cryptographic primitive is used across different operations, the outputs are distinguished by incorporating unique "context strings" during the computation, following the HMAC-based key derivation function approach outlined in [206].

Each authentication vector retrieved by the serving gateway consists of the following components:

- **RAND**: A random challenge used in the authentication process.
- **SEQ**: A sequence number that serves as an implicit challenge for the IoT device to authenticate the network, while also allowing the home authentication server to manage credential retrieval.

- **AUTN = HMAC$_{K_0}$(RAND | SEQ | "AUTN")**: An authentication token that enables the IoT device to verify the authenticity of the serving domain. The inclusion of the "AUTN" context string ensures distinct output for this specific purpose.
- **XRES = HMAC$_{K_0}$(RAND | SEQ | "XRES")**: The expected response from the IoT device to the gateway's authentication challenge. The use of a separate context string ("XRES") differentiates this output from other computations.
- **MKEY = HMAC$_{K_0}$(RAND | SEQ | "MKEY")**: The ephemeral master key, which is employed to generate session keys specific to the protocol.

A notable enhancement in our approach, compared to the 4G-AKA protocol, is the consistent use of both the random challenge (RAND) and the sequence number (SEQ) across all computations, including XRES—something not implemented in 4G-AKA due to backward compatibility constraints.

The mutual authentication handshake mirrors the steps of the AKA protocol used in cellular systems, with a few exceptions such as the non-prioritization of SEQ value anonymization in our implementation. In the handshake process, the IoT device first receives the RAND and SEQ values, recomputes the AUTN using its local secret key K_0, and verifies that the computed AUTN matches the one sent by the gateway, thereby confirming the network's authenticity. Subsequently, the device calculates the XRES value and transmits it back to the gateway, which then verifies that the received XRES matches the expected result stored in the authentication vector.

4.3.2 Key Agreement

A key distinction of our RIOT-AKA approach compared to the 4G AKA protocol is its flexibility to integrate with existing security protocols within RIOT OS. Instead of directly generating session keys, our approach derives an ephemeral master key (MKEY) during the RIOT-AKA handshake, which can then be used in conjunction with other established protocols. This design allows for enhanced adaptability and broader security applications.

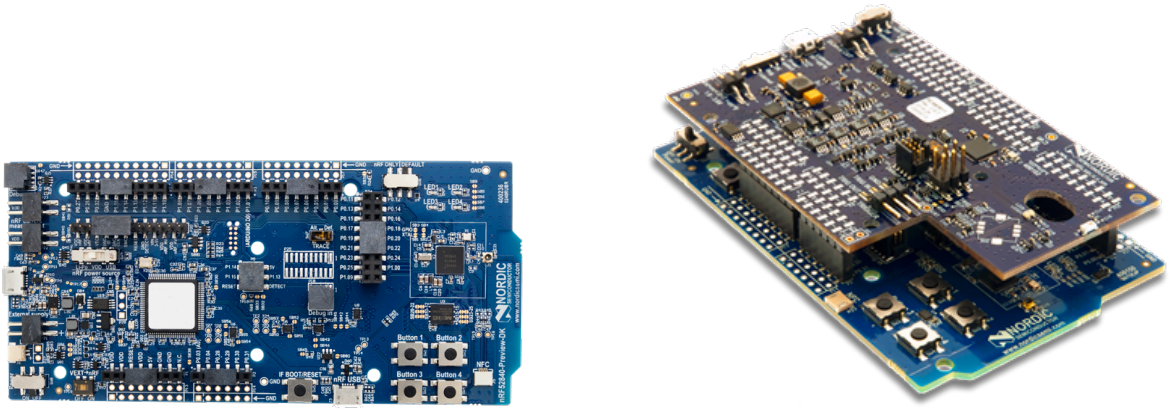
One potential integration involves using the derived MKEY as a pre-master key within the Datagram Transport Layer Security (DTLS) or Transport Layer Security (TLS) [207] stack. TLS and DTLS are widely adopted security protocols that provide robust protection against eavesdropping, tampering, and message forgery. Typically, these protocols generate pre-master keys through Ephemeral Diffie-Hellman (DHE) or Elliptic Curve Diffie-Hellman (ECDHE) exchanges, which are computationally intensive and resource-heavy, especially for constrained IoT devices [208]. Alternatively, TLS/DTLS can use Pre-Shared Keys (PSKs), but this method typically relies on static keys. By integrating RIOT-AKA, fresh keys can be dynamically generated with each authentication, allowing them to be used as PSKs in the TLS handshake, thus reducing computational overhead while enhancing security.

Another integration opportunity is leveraging the ephemeral MKEY within the Object Security for Constrained RESTful Environments (OSCORE) [209] protocol, which secures

application layer communications by protecting CoAP messages between endpoints. OSCORE encrypts and authenticates CoAP's payloads, including the request method, resource identifier, content format, and other sensitive fields, effectively safeguarding the integrity and confidentiality of IoT communications.

Both DTLS/TLS and OSCORE integrations are currently under development and are planned for inclusion in future versions of our prototype. These extensions aim to provide a more flexible and secure IoT environment by combining the dynamic key generation capabilities of RIOT-AKA with well-established security protocols.

4.4 Implementation and Extensions



(a) The nRF52840-DK board used.

(b) The power profiler mounted on top of the nRF board.

Figure 4.4: The nRF52840-DK board and the power profiler setup.

This section provides an overview of our experimental setup for implementing the RIOT-AKA protocol. Our primary hardware platform is the **nRF52840-DK**, an IoT device based on the Arm Cortex-M4 microcontroller from Nordic Semiconductor, featuring 1MB of Flash memory and 256kB of RAM. For reliable communication measurements, we use 6LoWPAN over Ethernet via serial connections. The experiments run on RIOT OS version 2021.01.

The setup consists of a RIOT node communicating with a virtual server acting as a "Gateway," implemented in Python, using a 6LoWPAN/CoAP protocol stack. The Gateway forwards requests via HTTP(S) to a "Provider" server, also implemented in Python, which supplies the necessary hashing materials for authenticating the IoT device. We develop a prototype to evaluate the execution flow of the RIOT-AKA protocol, with the implementation predominantly written in C. The code is open-source and available for reproducibility at [210] and [211].

For energy consumption measurements, we employ the **nRF6707** power profiling kit, specifically designed to work with the nRF52840-DK. Bandwidth measurements are conducted using Wireshark (version 3.2.3), focusing on data exchange between the IoT device and the Gateway. Memory usage is assessed using the open-source tool 'cosy' [212], which

analyzes .ELF files to provide a detailed breakdown of resource consumption across the entire protocol exchange, including network communication and hash generation. The software development on the nRF52840-DK utilizes the GNU Arm Embedded Toolchain (version 10-2020-q4-major) with GCC version 10.2.1.

4.4.1 SRAM-PUF for Secret Key Generation

To enhance our scheme, we explore the use of SRAM-PUFs (Physically Unclonable Functions) for generating secret keys, leveraging the unique hardware properties of IoT devices running on RIOT OS.

Background on PUFs

Physically Unclonable Functions (PUFs) utilize inherent physical variations in hardware, arising from manufacturing imperfections, to generate unique outputs that act as digital fingerprints [213][214]. These characteristics are used to produce random numbers and device-specific secret keys, which are valuable for authentication and securing data [215][216]. SRAM, commonly used in microcontrollers to temporarily store data, loses its stored values when powered off. Upon power restoration, each SRAM cell initializes with a random binary value, creating a distinct binary pattern known as an SRAM-PUF. However, due to the inherent noise in PUF outputs, helper-data algorithms, such as secure one-way hash functions, are necessary to extract stable and uniform keys. RIOT OS provides an API to facilitate the use of SRAM-PUFs for such purposes [217].

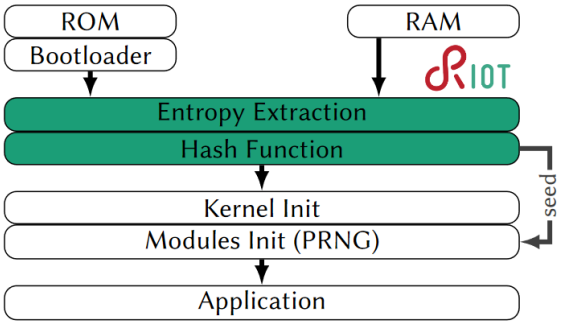


Figure 4.5: RIOT-OS PUF generation architecture.

Integration of SRAM-PUF in RIOT-AKA

As a proof-of-concept, we have integrated SRAM-PUF-based authentication within the RIOT-AKA framework. This feature enables the ”Provider” cloud domain to initiate a CoAP request to the IoT device, prompting the generation of a secret key using the SRAM-PUF. The RIOT module hooks into the OS startup routine, engaging right before kernel initialization. It processes uninitialized SRAM blocks through a hash function to derive a seed value [213]. Upon receiving the request, the IoT device responds with the newly generated key. It is important to note that this implementation is still experimental and not yet cryptographically secure. Future work will focus on addressing these limitations and exploring the viability of this approach in diverse use cases.

Table 4.1: Energy consumption

	PSK	SRAM-PUF	Diff
AVG (mA)	1.02	1.06	0.04
MAX (mA)	4.64	4.63	-0.01

Table 4.2: Bandwidth consumption

	PSK	SRAM-PUF	Diff
Linux CoAP (bytes)	149	149	0
RIOT-OS (bytes)	136	136	0
Time (s)	0.041	0.040	-0.001

Table 4.3: Memory consumption

	PSK	SRAM-PUF	Diff
Text (bytes)	70976	71152	176
Data (bytes)	832	832	0
Bss (bytes)	28196	28212	16
Tot (bytes)	100004	100196	192

Table 4.4: ROM and RAM consumption

	PSK	SRAM-PUF	Diff
ROM (bytes)	71808	71984	176
RAM (bytes)	29028	29044	16

Pros and Cons of Using SRAM-PUF in RIOT-AKA

SRAM-PUFs offer distinct advantages for IoT devices operating in resource-constrained environments, where traditional security mechanisms may be impractical. They provide a low-cost alternative for cryptographic applications, especially when devices are susceptible to capture or tampering. Unlike conventional methods that store keys in nonvolatile memories (NVMs) or battery-backed RAMs—both of which are vulnerable to invasive attacks—SRAM-PUFs generate volatile values that are not permanently stored, enhancing security. However, this volatility also poses challenges; for example, hard-resetting the device (i.e., power cycling) necessitates renegotiating keys with the ”Provider” cloud domain and reinitializing all authentication phases, which could introduce vulnerabilities in uncontrolled environments.

4.5 Tests and Results

This section presents the results collected from our experiments, with a focus on the IoT device, given its hardware and software constraints which make it the most critical component of the prototype. We compare the performance of two key generation modes: Pre-Shared Key (PSK) and SRAM-PUF. This comparison is not intended to benchmark the two implementations directly, as they largely share the same code base, but rather to provide an overview

of how each mode performs on actual hardware. In the PSK mode, a 32-byte key is used, while the SRAM-PUF mode generates a 10-byte key from the PUF. The keys are assumed to be readily available or generated prior to the tests, thus the key-provisioning phase is not included in the evaluation. Measurements begin with the transmission of the initial authentication packet and conclude when the final packet is sent.

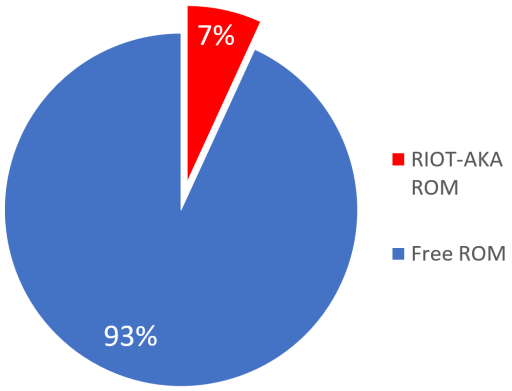
4.5.1 Energy Consumption Impact

Table 4.1 shows a 3.9% increase in average energy consumption when switching from PSK to SRAM-PUF key generation. This difference is considered negligible and likely due to normal execution variations within the narrow measurement window, rather than a specific impact of the key generation mode. The peak energy consumption also shows minimal variation, with a 0.22% decrease observed when using SRAM-PUF compared to PSK.

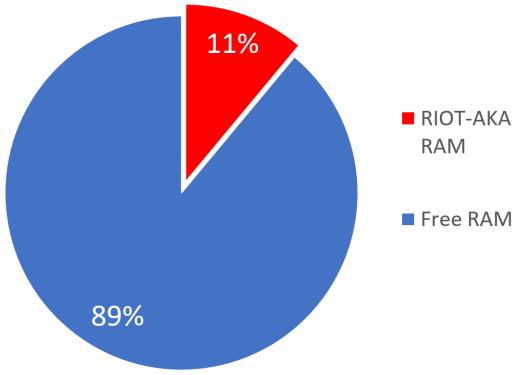
4.5.2 Bandwidth Consumption Impact

As shown in Table 4.2, there is no noticeable difference in the amount of data transmitted over the network between the PSK and SRAM-PUF implementations. Both modes function identically in terms of communication, differing only in the key used for HMACs. The time taken for communication is nearly identical, showing only a 2% reduction in the SRAM-PUF mode, which is attributed to the minor variations in processing.

4.5.3 Memory Consumption Impact



(a) ROM portions used by our implementation.



(b) RAM portions used by our implementation.

Memory usage, detailed in Tables 4.3 and 4.4, shows slight differences between the two implementations due to the inclusion or exclusion of SRAM-PUF modules in the compilation process, controlled by a Makefile flag. The 'Text', 'Data', and 'Bss' sections refer to segments within the compiled .ELF files: 'Text' represents the code stored in ROM (flash memory), 'Data' includes initialized variables that reside in both ROM and RAM, and 'Bss' contains uninitialized variables stored in RAM. Specifically, using SRAM-PUF results in a

0.25% increase in ROM usage (176 bytes) and a 0.06% increase in RAM usage (16 bytes) compared to PSK. Figures 4.6a and 4.6b visually depict the ROM and RAM allocations on the nRF board, illustrating that the differences between the two modes are minimal. Overall, the memory footprint of our prototype is well within acceptable limits for IoT environments, and further optimizations could potentially reduce this even more.

4.6 Summary

This chapter demonstrates that well-established authentication mechanisms, long utilized in the cellular networking domain, can be effectively adapted for IoT devices. These mechanisms, known for their simple and robust roaming models, align well with the dynamic nature of IoT deployments, where devices may frequently relocate or transition between different smart spaces. From a technical standpoint, the deployment on IoT platforms is straightforward, as evidenced by our implementation on RIOT OS using CoAP. By substituting the specialized cryptographic algorithms commonly employed in cellular systems with widely available HMAC and HKDF constructions, we achieve a practical and accessible solution for IoT environments.

However, we recognize that this preliminary proof-of-concept is only a starting point, and several critical aspects remain to be addressed to develop a comprehensive and fully robust implementation. Our ongoing work is focused on two primary objectives: (i) enhancing and extending the protocol with improved registration and management processes, including mechanisms to handle desynchronization events and potential attacks, and (ii) integrating the protocol with established security protocols, such as (D)TLS and OSCORE, which are already supported by RIOT OS. These efforts aim to strengthen the overall security framework and ensure the scalability and resilience of our approach in real-world IoT applications.

Chapter 5

Adversarial Machine Learning As A New Threat to AI-Driven Systems

5.1 Introduction

In the previous chapters, we explored how the application of Machine Learning (ML) algorithms to Internet of Things (IoT) data and simulated data in Smart Water Distribution Networks (SWDNs) enhances their capabilities in detecting leakages and malfunctions. Numerous studies have utilized image-based methods too to detect leaks in Water Distribution System (WDS) [153].

IR thermography has been used since the 1980s to identify leaks in buried pipes and detect surrounding erosion [218, 219, 220]. Recent approaches have integrated IR thermography with ground-penetrating radar (GPR) for improved accuracy, using techniques like region-growing algorithms and Green's theorem for more precise localization [221]. Other methods, such as visible light imaging and Otsu thresholding, have also been employed to enhance leak classification [222, 223, 224].

In addition to imaging techniques, audio and vibration-based signals have been explored for leak detection. Feature extraction from these signals, followed by ML classification, has shown promising results. Hybrid models, combining convolutional neural networks (CNNs) and Support Vector Machines (SVMs), have demonstrated strong performance in these tasks [225, 226, 227]. Studies consistently show that ML models outperform traditional methods in both accuracy and speed [228, 226, 229, 230].

As reliance on ML and artificial intelligence (AI)-based frameworks increases, so do the associated security risks. Adversarial machine learning (AML) poses a notable threat, as even small, imperceptible perturbations in data—particularly images—can lead to misclassifications. While initially explored in computer vision, this vulnerability has real-world implications across various domains, including Water Distribution Networks (WDNs), where ML-driven leak detection relies heavily on image-based techniques like IR thermography and GPR. In adversarial scenarios, malicious actors could exploit these weaknesses to disrupt leak detection and other AI-driven analyses, ultimately compromising the reliability of

critical water infrastructure.

Although the application of AML to WDNs remains largely theoretical, it highlights the urgent need for robust ML frameworks capable of *understanding* adversarial manipulation. As ML becomes increasingly integrated into real-world systems, such as WDNs, ensuring the integrity of these models will be crucial for protecting critical infrastructure from both natural failures and malicious attacks.

5.2 Semantic Subgraph Extraction Attacks To Convolutional Neural Networks

CNNs have been shown to be vulnerable to adversarial perturbations [231, 232, 233]. Although AML research primarily focuses on developing methods to create these perturbations, it often requires substantial computational power. These demands can vary considerably based on the specific CNN task, dataset size, model complexity, and the desired performance outcomes [234, 235]. For instance, while advanced AML attacks such as Carlini & Wagner (C&W) [236] can attain higher success rates—sometimes reaching 100% [237]—compared to simpler attacks like Projected Gradient Descent (PGD) [238] and Fast Gradient Sign Method (FGSM) [239], C&W can be as much as 13,000 times slower [240]. *This computational burden underscores a gap between existing AML methods and their practicality in real-world applications.*

This issue becomes more pronounced when CNNs are updated over time through techniques like fine-tuning [241], few-shot learning [242], or test-time adaptation [243]. As CNNs grow in complexity, the latency and computational costs associated with AML techniques are expected to rise accordingly [244].

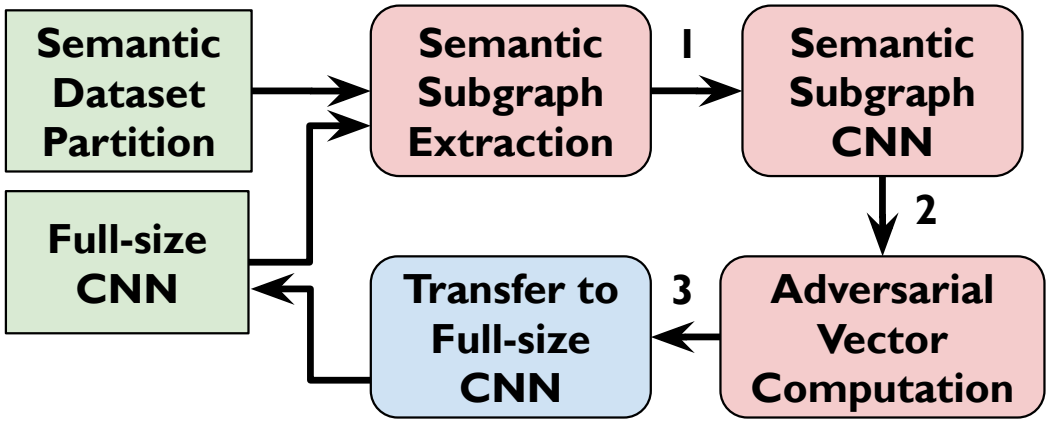


Figure 5.1: Overview of SSEA. We first extract a subgraph CNN from the full-size CNN evaluating a semantic cluster of the dataset (1), we execute an existing white-box attack on the subgraph (2), and we transfer the adversarial vector to the full-size CNN (3).

To address these challenges, we introduce a novel AML approach, *semantic subgraph extraction attack (SSEA)*, depicted in Figure 5.1. Our method builds on the understanding that CNNs effectively capture semantic correlations between spatial features in high-dimensional

data [245]. Specifically, semantic similarities—such as shared attributes or meanings between classes—can play a crucial role in shaping both the attack strategy and its computational cost.

We test this hypothesis by applying our approach to widely-used CNNs, including ResNet-50, VGG-16, and VGG-19, using the C&W L_2 attack on the CIFAR-100 dataset and selected subsets of Tiny-Imagenet. These datasets are standard benchmarks in AML research for evaluating adversarial attacks in computer vision [246]. Our experiments demonstrate that targeting classes with similar semantic features reduces input perturbations by up to 41% for CIFAR-100 and 38% for Tiny-Imagenet, while also cutting down the number of required iterations by up to 23% and 10%, respectively.

We incorporate insights from [247], which demonstrate how CNNs can be reduced to smaller surrogate CNNs that focus on a semantic subset of classes without the need for fine-tuning. Our method specifically targets this subset, referred to as a "*subgraph*," instead of operating on the entire network. Using these reduced-complexity CNNs, we generate adversarial inputs, which are then applied to the full-sized CNNs, effectively simulating attacks as though they were crafted on the original network, as depicted in Figure 5.1. This approach, SSEA, integrates seamlessly with any white-box attack, drastically reducing the computational complexity of adversarial attacks while preserving their effectiveness. In fact, SSEA decreases the floating point operations (FLOPs) and parameter counts by up to 21% and 44%, respectively, for CIFAR-100, with similar improvements observed for Tiny-Imagenet.

To assess the efficacy of adversarial inputs generated on the subgraph in fooling the full-sized model, we compare the predictions of the full-sized CNN using adversarial inputs from both the subgraph and the original model. This comparison evaluates whether the subgraph-derived inputs successfully mislead the full-size model to misclassify a specific target adversarial class. We compute the attack success rate (ASR) for both sets of inputs, using the full-sized model's predictions to measure any ASR loss during the transfer process. A minimal loss would indicate that the subgraph preserves the key characteristics and vulnerabilities of the original CNN, ensuring effective attack transfer.

In particular, when testing transferability with PGD and MIFGSM attacks on CIFAR-100, the top-1 ASR decrease for VGG-19 is only 3.63% and 6.56%, respectively. Similarly, for Tiny-Imagenet, the top-3 ASR decrease for VGG-16 under PGD and MIFGSM attacks is 7.28% and 8.66%, respectively. These minor reductions confirm the effectiveness and transferability of our approach. In certain cases, adversarial inputs generated from subgraphs even surpass those crafted directly on the full-sized CNNs. **Furthermore, SSEA introduces negligible additional latency, as it requires no further fine-tuning, while providing new insights into CNNs behavior, representing a novel advancement in improving AML performance.**

5.3 Background and intuition

We remind that an *adversarial input* is created by deliberately modifying an input so that its representation is shifted into an incorrect classification region within the model's decision space [248]. This process alters the input data x by adding a noise vector δ, referred to as a *perturbation*, in order to induce classification errors in the neural network. These adversarial inputs are typically constrained by a predefined maximum distance, and various distance metrics are used to assess the perturbation's effectiveness. These metrics include sparsity (L_0), total extent (L_1), energy (L_2), and maximum change (L_∞) [249, 250, 251, 236, 239]. Attacks are generally categorized as either *targeted*, where the input is forced into a specific incorrect class, or *untargeted*, where the goal is to misclassify the input in any incorrect class. When dealing with RGB images, the input data x is structured as a three-dimensional matrix,

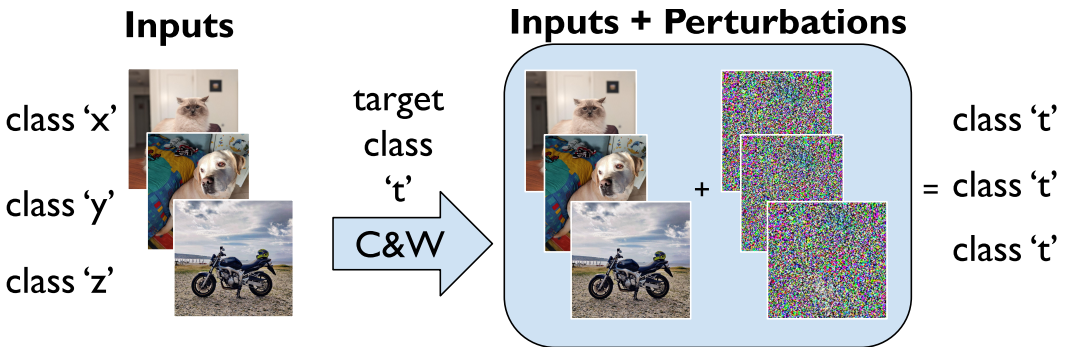


Figure 5.2: Targeted C&W attack on images from classes 'x', 'y', and 'z'. Specific perturbations are added to each image, creating adversarial examples that are all misclassified as the target class 't'.

representing the image's color channels with dimensions $3 \times h \times w$. As illustrated in Figure 5.2, adversarial perturbations specific to image classes x, y, and z result in misclassification into class t. We apply the C&W L_2 attack [236], aiming to minimize the perturbation δ while ensuring that $C(x+\delta) = t$:

$$\begin{aligned} &\min_{\delta} ||\delta||_p + c \cdot f(x+\delta) \\ &\text{s.t.} \quad x + \delta \in [0,1]^{3 \cdot h \cdot w} \end{aligned} \tag{5.1}$$

Here, $||\delta||_p$ represents the L_2 norm of the perturbation, which measures its magnitude. The term $f(x+\delta)$ is often calculated using cross-entropy loss, where $f(x+\delta) \leq 0$ indicates $C(x+\delta) = t$. The parameter c controls the balance between minimizing the perturbation and achieving the classification objective. This attack utilizes iterative gradient descent, adjusting c based on whether $C(x+\delta_c) = t$, and repeats for $n_g \cdot n_c$ iterations, where n_c is the number of binary search steps and n_g represents iterations per binary search. The smallest perturbation δ that achieves $C(x+\delta_c) = t$ is chosen as the final adversarial input; if none are successful, the attack fails.

5.3.1 Related work

There has been limited exploration into modifying CNN architectures to speed up attacks, reduce model complexity, or use surrogate models for more efficient execution.

In [252], a Meta-Transfer Attack (MTA) framework is introduced, which uses a Meta-Surrogate Model (MSM) to generate adversarial examples with high transferability. The MSM is trained to enhance the transfer of these examples to various target models, improving overall attack efficiency. However, this approach requires significant trial and error and extensive fine-tuning, and primarily focuses on black-box scenarios. In [253], a "Fast C&W" algorithm is proposed, which uses a deep encoder network to efficiently produce adversarial examples for SAR target recognition systems. This method accelerates the attack while maintaining high effectiveness but is limited to the C&W attack and is not generalized for other input types or tasks. Similarly, [254] investigate the use of skip connections to improve the transferability of adversarial inputs between models, but their approach does not address the acceleration or simplification of the attack process. In another study, [248] propose refining adversarial examples to improve transferability across models, but they add a fine-tuning step without emphasizing attack speed. Additional methods for faster or optimized attacks have been suggested by [255], [256], and [257], though these studies typically focus on individual attack methods and do not examine the broader optimization of multiple attacks or the reduction of model complexity.

Furthermore, a variety of neural network manipulation strategies have been explored as defensive mechanisms [258, 259, 260, 261, 262, 263, 264], but their potential application for enhancing attack efficiency remains largely unexamined.

5.3.2 Impact of Semantic Similarity on Attack Performance

We investigate whether leveraging the semantic properties of class groupings can enhance adversarial attack efficiency. To validate this hypothesis, we employ the C&W attack as a benchmark.

The performance of a successful C&W attack is measured by the total iterations $N(x) \leq n_g \cdot n_c$ required to discover the optimal perturbation. We evaluate VGG-16, VGG-19 [265], and ResNet-50 [266] CNNs using the CIFAR-100 [267] and Tiny-Imagenet [268] datasets. While these models and datasets are not the most recent, they remain popular and comprehensive benchmarks in ML research [269], serving as a robust foundation for our study. CIFAR-100 comprises 100 *fine* classes grouped into 20 *coarse* classes, with each coarse class containing five semantically related fine classes. For example, the "aquatic mammals" coarse class consists of the fine classes "beaver," "dolphin," "otter," "seal," and "whale." Conversely, Tiny-Imagenet lacks an inherent fine-to-coarse class hierarchy. To address this, we manually categorized 25 of Tiny-Imagenet's fine classes into 5 coarse groups based on semantic and visual similarities. We refer to this modified dataset as "Tiny-HL" (Hierarchical Labeling). To ensure compatibility with models pre-trained on Imagenet, Tiny-HL images

were upscaled to 224x224 pixels. Imagenet itself was excluded due to hardware and time constraints.

Our hypothesis is that intra-class attacks—targeting fine classes within the same coarse class—require smaller perturbations and improve attack performance compared to inter-class attacks, which target fine classes from different coarse classes.

To maintain consistency in our evaluations, we filtered the dataset D to include only images that the models correctly predicted. All subsequent analysis is based solely on these correctly classified images.

We conducted multiple attacks on each image in the dataset, transitioning each input from its original fine class to all other target fine classes within the dataset. The C&W attack was configured with a learning rate of 5×10^{-3}, a confidence level of 0, and a perturbation bound within the range of the original dataset after standard normalization. The attack parameters included $n_c = 5$ binary search steps and $n_g = 1000$ iterations per step.

Table 5.1: Adversarial Attack avg total iterations for CIFAR-100

CNN	**Type**	$E_{D'}[N]$	**% Faster**
VGG-16	Inter-class	2037.78	22.88%
	Intra-class	1571.44	
VGG-19	Inter-class	2010.13	18.29%
	Intra-class	1642.53	
ResNet-50	Inter-class	1617.12	20.63%
	Intra-class	1283.58	

Table 5.2: Adversarial Attack avg total iterations for Tiny-HL

CNN	**Type**	$E_{D'}[N]$	**% Faster**
VGG-16	Inter-class	2365.98	7.12%
	Intra-class	2197.45	
VGG-19	Inter-class	2558.09	8.18%
	Intra-class	2348.83	
ResNet-50	Inter-class	2858.18	9.79%
	Intra-class	2578.30	

Tables 5.1 and 5.2 summarize the average total iterations $E_{D'}[N]$ needed to generate the optimal adversarial sample across all images in D'. Additionally, Figure 5.3 illustrates the mean$\pm$std L_2 perturbations for different datasets and models. A clear trend emerges: **intra-class attacks consistently exhibit greater efficiency compared to inter-class attacks, both in terms of convergence time and perturbation magnitude**. For Tiny-HL, intra-class attacks lead to up to 9.79% faster mean convergence times across models, along with a reduction in mean L_2 perturbation sizes by as much as 38.32%. Similar results are observed for CIFAR-100, where intra-class attacks achieve up to 22.88% faster mean convergence and reduce perturbation sizes by up to 50.85%.

Figure 5.3 presents two subsets of adversarial attacks on the ResNet model using the CIFAR-100 dataset. These subsets focus on transitions where images shift from their original

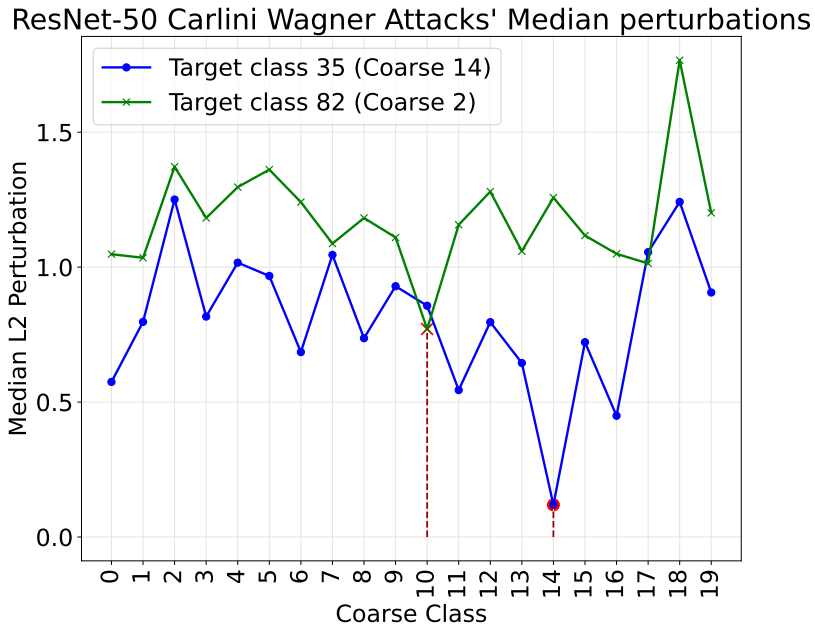


Figure 5.3: ResNet-50 easiest and hardest target attack between coarse classes. Lower values mean less perturbation to the original input.

fine class to a target fine class in a different coarse class. Specifically, we target fine label 82 within coarse label 2 and fine label 35 within coarse label 14. The y-axis denotes the perturbation magnitude, while the x-axis represents the original coarse class of the images. The median perturbation magnitude is calculated for each sample set.

The blue trajectory shows that images originating from coarse class 14, which aligns with the target class's coarse class, tend to have lower perturbation magnitudes. This subset also exhibits the smallest median perturbation among the attacks where the original and target coarse classes match. However, this pattern is not consistent across all scenarios. The green trajectory reveals cases where minimal perturbation magnitudes arise from transitions between different source and target coarse classes, highlighting the complex and varied nature of adversarial perturbations.

5.4 Our Approach

Building on prior analyses, we observed that attacks targeting classes within the same coarse group often require less computational effort. The concept of *semantic similarity* refers to the extent to which two classes share common attributes, meanings, or contexts. Classes are considered semantically similar when they possess related meanings, a concept that applies across various domains, including image classification and natural language processing [270].

We combine our intuition with findings from [247], which show that semantically similar inputs tend to activate a higher number of common filters, particularly in the early layers of a CNN. For example, images of birds like sparrows and eagles activate more similar filters than images of birds and automobiles. Additionally, in some scenarios, CNN inputs may

Algorithm 1 Semantic Subgraph Extraction Attack (SSEA)

Input: Neural Network G, Dataset Subset D_s, Extraction Rates **er**, Target Class $t \in D_s$
Output: Extracted Subnetwork G_s, Adversarial Inputs X_{adv}
Initialize $A_{\text{sq}} \leftarrow 0$ for each layer l
for each batch b in D_s **do**
  Perform inference on b using G
  Update $A_{\text{sq}}[l] \leftarrow A_{\text{sq}}[l] + (A_b^{(l)})^2$ for each layer l
end for
for each layer l **do**
  Compute $L2^{(l)} \leftarrow \sqrt{A_{\text{sq}}[l]}$
  Remove bottom er_l
end for
Adjust output layer size of G_s based on D_s
$G_s \leftarrow$ Remaining structure of G
Craft adversarial inputs X_{adv} using G_s, targeting class t
Transfer X_{adv} against full-size network G
return G_s, X_{adv}

be limited to a set of semantically similar classes, which often exhibit strong temporal correlations [247]. As a result, semantically similar classes can be clustered together to form smaller, class-specific models. Unlike traditional structured pruning techniques [271], this approach does not require fine-tuning and retains the original weights, preserving most of the full-size model's receptive field and saliency maps.

Following this principle, our SSEA approach targets CNNs by focusing on a subset of semantically similar classes within one or two coarse groups. To optimize these attacks, we extract relevant subgraphs from the CNNs and craft adversarial inputs on these smaller subgraphs, which reduces the computational complexity. Our approach does not seek to replace existing techniques, but rather to enhance them by integrating any existing white-box attack into an adaptive framework.

Moreover, rather than directly optimizing the attacks, we adjust the CNN architecture based on the input and target adversarial classes. This approach minimizes deviations from the original attack direction when applied to the full-size model, ensuring that the adversarial inputs remain effective when transferred back to the original network. Significant deviations can undermine the success of the attack [248], which our method seeks to avoid.

5.4.1 Understanding SSEA's process

As depicted in Figure 5.4 and outlined in Algorithm 1, our subgraph extraction process begins by running inference on a subset of the dataset, D_s, containing samples from all fine-grained classes within the targeted and source coarse classes. Let G represent the CNN's graph structure, with its corresponding weights W.

During the inference on D_s, the activation values of each neuron in layer l are denoted by the tensor $A_b^{(l)}$ for each batch b. The cumulative activation for each neuron is calculated by summing the squared activations across all batches:

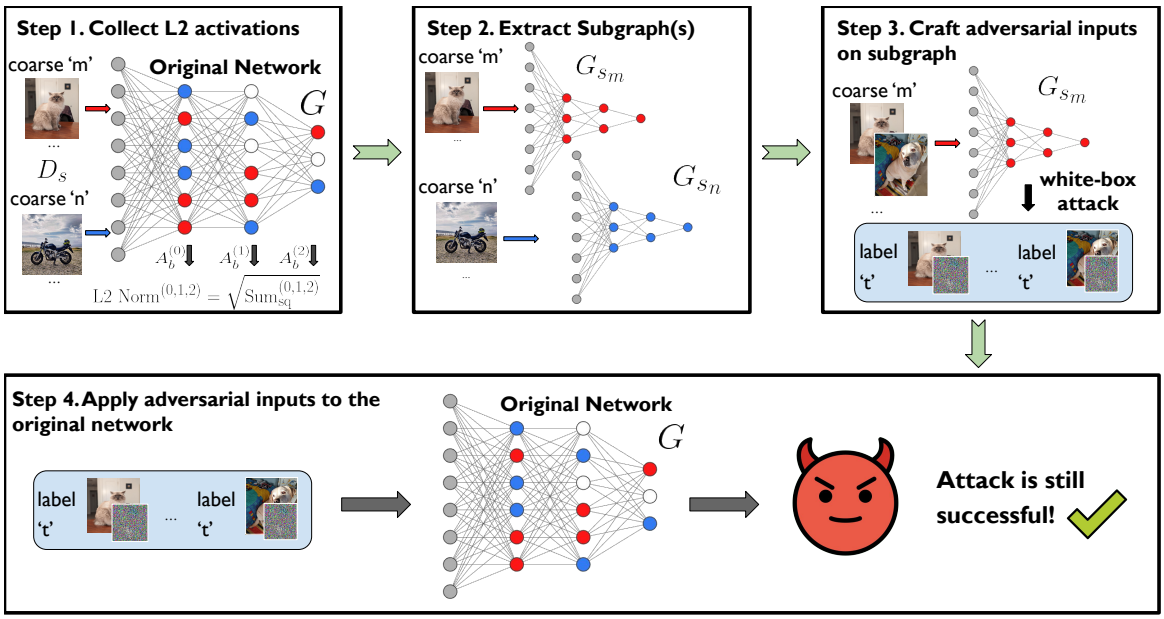


Figure 5.4: Simplified representation of all SSEA operations.

$$\text{Sum}_{\text{sq}}^{(l)} = \sum_{b=1}^{B} \left(A_b^{(l)} \right)^2$$

where B represents the total number of batches. The L2 norm for each layer l is then computed as:

$$\text{L2 Norm}^{(l)} = \sqrt{\sum_{j=1}^{m_l} \text{Sum}_{\text{sq},j}^{(l)}}$$

Neurons or filters with lower L2 norms are considered less important and are selected for removal, based on the reduction percentage $\mathbf{er} = [er_1, er_2, \ldots, er_L]$, where each er_l specifies the proportion of neurons or filters to retain in layer l.

The output layer of the extracted subgraph is adjusted according to the targeted coarse classes. For subgraphs targeting two coarse classes, the output layer consists of 10 classes, corresponding to the fine classes within those two coarse groups. For subgraphs targeting a single coarse class, the output layer is limited to 5 classes, denoted as C_t.

The final subgraph, G_s, is a reduced version of G, with a subset of its weights W_s, such that:

$$G_s(W_s) = E(G(W), D_s, \mathbf{er})$$

The weights W_s remain identical to their counterparts in the original model, ensuring that the learned representations are preserved in G_s. This subgraph is then used to generate adversarial inputs, X_{adv}, using any white-box attack technique. These adversarial inputs are subsequently transferred to the full-size CNN, thereby simulating an attack as if it were directly crafted on the original network. The goal is to ensure that X_{adv} is effective, meaning

that the original full-size CNN misclassifies these adversarial inputs into the target class t. The attack is considered successful if the output f of G satisfies $f(X_{\text{adv}}) = t$.

Table 5.3: Comparison of FLOPs and number of parameters between original (Org.) models and subgraphs (Sub.) using SSEA on CIFAR-100 and Tiny-HL. Smaller values are better.

Model	Dataset	Org. FLOPs	Sub. FLOPs	Org. Params	Sub. Params
VGG-16	CIFAR-100	314M	**269M**	14.8M	**9.2M**
VGG-19	CIFAR-100	399M	**317M**	20.1M	**11.3M**
ResNet-50	CIFAR-100	1.31B	**1.12B**	23.7M	**20.0M**
VGG-16	Tiny-HL	314M	**306M**	14.8M	**12.6M**
VGG-19	Tiny-HL	399M	**354M**	20.1M	**15.9M**
ResNet-50	Tiny-HL	84.7M	**76.6M**	23.9M	**21.3M**

5.5 Experimental Results

5.5.1 Setup

Our experiments are conducted using the PyTorch deep learning library [272]. The attacks are implemented using the TorchAttacks library [273] and the CleverHans [274] implementation of the Carlini-Wagner L_2 attack. Our subgraph extraction framework builds on the Torch-Pruning library [275], with custom code added to extend its functionality. All experiments are performed in a Fedora-based Linux environment with 512 GB of RAM and Nvidia A100 80GB GPUs.

The adversarial attacks evaluated in this section include FGSM L_∞ [239], PGD L_∞ [238], BIM L_∞ [276], MIFGSM L_∞ [277], Jitter L_∞ [278], and EOTPGD L_∞ [279]. For further details on model configurations, training procedures, dataset cleaning, subgraph extraction rates, and extended results, a technical appendix is provided. Additionally, the codebase and trained model weights will be released as open-source resources after the research is completed.

5.5.2 Comparison of CNN Complexity Reduction

We evaluate the computational cost of crafting adversarial inputs using the metric of FLOPs (floating-point operations per second), which depends on the size, architecture, and complexity of the CNNs. This hardware-independent metric provides a clear measure of model complexity [280] and its impact on memory usage. Specifically, FLOPs quantify the operations required for a single forward pass through the model, while in AML contexts, they represent the operations needed for forward and backward passes to generate adversarial examples.

It is important to note that reducing model parameters does not necessarily result in a proportional reduction in FLOPs, as the overall effect depends on the architecture and the complexity of individual layers.

Table 5.3 compares the reduction in FLOPs and model size relative to the baseline models within the attacked subgraphs. For CIFAR-100, VGG-16 and VGG-19 models show reductions in FLOPs of up to 20.66%, while ResNet-50 achieves a reduction of 14.97%. On the Tiny-Imagenet dataset, FLOPs are reduced by up to 11.37%. Regarding model size, our approach results in parameter reductions of up to 43.82% for VGG-16 and VGG-19 on CIFAR-100, while ResNet-50 achieves a 15.76% reduction. On Tiny-Imagenet, models like VGG-19 and ResNet-50 show parameter reductions of up to 21.0%.

5.5.3 SSEA ASR Loss Evaluation after Transfer

We evaluate SSEA using the ASR as the primary success metric. ASR quantifies the percentage of adversarial attacks that successfully mislead a neural network into making incorrect predictions. Additionally, Top$_k$ ASR measures the percentage of adversarial attacks that exclude the true label from the model's Top$_k$ predictions, instead assigning the target adversarial class.

In the context of AML, transferability refers to the capacity of adversarial inputs generated for one model to induce misclassifications in another model. This occurs when adversarial inputs cause a victim model to misclassify inputs into a targeted adversarial class. In our study, transferability is localized, as both the source and target models share the same weights. Generally, when transferring adversarial inputs, a decrease in ASR is expected. However, a minimal ASR decrease—close to zero—suggests that the adversarial inputs remain effective, indicating that the subgraph effectively identifies the critical components of the full-size model that respond to both source and target fine labels within two semantic groups.

In this evaluation, adversarial inputs are generated using both subgraph-extracted CNNs and the full-size model. We then compare the predictions of the full-size CNN when using adversarial inputs generated from subgraphs with those crafted directly on the full-size model. The ASR for both sets of inputs is computed on the full-size model, and the ASR decrease (denoted as ASR$_{loss}$) during the transfer is recorded.

Table 5.4: Average Top$_k$ ASR decrease between original and extracted model attacks (CIFAR-100, $\epsilon = 0.1$, steps = 10). Values near zero indicate similar performance to the full-size model, while negative values suggest better subgraph performance.

Model	ASR$_{loss}$	PGD	BIM	FGSM	MIFGSM	EOTPGD	Jitter
VGG-16	Top$_1$	5.02%	5.02%	-0.09%	7.21%	5.02%	-1.81%
	Top$_3$	-2.74%	-2.75%	-0.09%	1.18%	-2.74%	-8.63%
	Top$_5$	-3.45%	-3.45%	-0.05%	1.00%	-3.45%	-12.18%
VGG-19	Top$_1$	3.63%	3.63%	0.03%	6.56%	3.63%	-1.79%
	Top$_3$	-2.25%	-2.25%	0.32%	2.76%	-2.25%	-8.96%
	Top$_5$	-3.05%	-3.05%	0.59%	2.53%	-3.05%	-12.78%
ResNet-50	Top$_1$	28.25%	28.25%	-0.20%	22.85%	28.25%	-0.99%
	Top$_3$	19.70%	19.70%	-0.47%	16.99%	19.70%	-8.62%
	Top$_5$	16.76%	16.76%	-0.48%	14.86%	16.76%	-12.01%

Table 5.5: Average Top_k ASR decrease between original and extracted model attacks (Tiny-HL, $\epsilon = 0.03$, steps = 5). Values near zero indicate similar performance to the full-size model, while negative values suggest better subgraph performance.

Model	ASR_{loss}	PGD	BIM	FGSM	MIFGSM	EOTPGD	Jitter
VGG-16	Top_1	19.47%	19.48%	5.51%	20.44%	19.48%	-11.13%
	Top_3	7.28%	7.28%	5.38%	8.66%	7.28%	-16.96%
	Top_5	4.23%	4.23%	4.63%	5.49%	4.23%	-19.40%
VGG-19	Top_1	38.78%	38.78%	6.19%	37.75%	38.78%	-5.54%
	Top_3	22.32%	22.32%	7.60%	24.32%	22.32%	-12.41%
	Top_5	16.37%	16.37%	7.26%	18.61%	16.37%	-14.84%
ResNet-50	Top_1	60.44%	60.44%	0.39%	49.34%	60.44%	-2.99%
	Top_3	52.97%	52.97%	-0.46%	44.42%	52.97%	-10.44%
	Top_5	48.93%	48.93%	-0.90%	41.75%	48.93%	-13.29%

In our SSEA CIFAR-100 evaluation (Table 5.4), VGG-16 and VGG-19 exhibit minimal ASR_{loss} and align closely with the full-size model, while ResNet-50 demonstrates greater divergence. Specifically, VGG-16 shows a slight improvement in Top_1 ASR over the full-size model under FGSM (-0.09%) and Jitter (-1.81%), with further improvements in Top_3 and Top_5, indicating superior subgraph performance. VGG-19 experiences minimal ASR decrease, with Top_1 reductions of 3.63% under PGD and 6.56% under MIFGSM, while performing better at higher ranks. In contrast, ResNet-50 suffers a notable Top_1 ASR decrease under PGD (28.25%) and MIFGSM (22.85%), but performs better at Top_5 with FGSM (-0.48%). Jitter attacks generally enhance performance across all models.

For Tiny-HL (Table 5.5), VGG-16 and VGG-19 exhibit improved performance under Jitter attacks at all ranks. VGG-16 outperforms the full-size model with Top_1 (-11.13%), Top_3 (-16.96%), and Top_5 (-19.40%) ASR improvements, while VGG-19 demonstrates similar trends. On the other hand, ResNet-50 shows significant ASR decreases under PGD (60.44%) and MIFGSM (49.34%), but improves under Jitter at Top_1 (-2.99%), Top_3 (-10.44%), and Top_5 (-13.29%).

These substantial deviations suggest that the extraction process may not fully preserve critical features in more complex architectures like ResNet-50. This highlights the necessity for refined metrics that better capture the original CNN's functionality in its subgraph form. Additionally, different attack types exhibit varied behavior due to the structural differences between CNNs. Iterative attacks like PGD exploit deeper vulnerabilities in the model's parameter space, making the extracted CNNs more sensitive to these variations. In contrast, attacks like FGSM and Jitter, which are less fine-grained, may not fully capture the changes caused by structural modifications in the extracted CNNs.

5.6 Summary

This study introduces a novel approach in AML that accelerates attacks on CNNs by modifying their architectures based on the semantic relationships between target classes. We

hypothesized that targeted attacks between semantically similar classes would enhance attack performance. Our results confirm this hypothesis, demonstrating up to 22.88% faster convergence and a reduction of up to 40.45% in mean perturbation size for CIFAR-100, with similar outcomes for Tiny-HL. Building on this insight, our key contribution is the development of a subgraph extraction-based attack, named SSEA, which isolates portions of the full-size CNNs that are responsive only to specific groups of semantically similar classes.

SSEA achieves reductions in FLOPs of up to 20.66% on CIFAR-100 and up to 9.57% on Tiny-HL. Parameter reductions are even more significant, reaching up to 43.82% for CIFAR-100 and up to 21% for Tiny-HL. Moreover, we show that these subgraphs retain high transferability and, in many cases, outperform the original CNNs, providing practical solutions to AML challenges and offering deeper insights into the vulnerabilities of CNNs.

In future work, we plan to extend our methods to newer architectures, including transformers. While our current approach is tailored to CNNs, applying similar techniques to transformers presents unique challenges that require further exploration. We intend to adapt and refine our methods to investigate the specific vulnerabilities of these emerging models.

Additionally, this research could lead to new benchmarks for evaluating CNNs's robustness, measuring their resilience not only to traditional adversarial attacks but also to subgraph manipulations. Such benchmarks would contribute to enhancing the security and reliability of these architectures. Understanding how to manipulate CNNs at a fundamental level will open avenues for new defense mechanisms, CNN optimization strategies, and a deeper understanding of this evolving field.

Chapter 6

Conclusions

This thesis addresses several technical and theoretical challenges in developing sustainable, modern applications for critical infrastructures, with a specific focus on Water Distribution Networks (WDNs). We propose solutions that leverage Internet of Things (IoT), Machine Learning (ML), and Low Power Wide Area Network (LPWAN) technologies, specifically Long Range Wide Area Network (LoRaWAN), to tackle environmental concerns and optimize the management of sensitive water systems.

In the first part of the work, we focus on IoT-based solutions for improving sustainability in WDNs. We emphasize the integration of IoT, edge computing, and ML within these networks, showcasing their potential to optimize system performance by moving computations to the edge. Strategic placement of gateways in LoRaWAN networks plays a crucial role, as optimal positioning directly impacts data quality and overall efficiency. Our simulations demonstrate how variations in network characteristics affect data extraction rates, providing insights for effective deployment strategies.

We also evaluate several ML models, such as Multilayer Perceptrons (MLPs), convolutional neural networks (CNNs), and Graph Neural Networks (GNNs), in predicting pressure values using minimal input features. Each model demonstrates varying levels of effectiveness depending on the complexity of the network. Introducing random leakages further highlights these models' limitations, emphasizing the need for more resilient leak detection approaches.

Building on these insights, we present SWIM, an integrated framework that combines IoT, simulations, Digital Twins (DTWs), and ML to simplify the complex management of WDNs. SWIM offers a comprehensive platform for managing these systems through IoT-enabled sensors, real-time data analysis, and ML-based predictions. By incorporating DTWs, SWIM facilitates real-time monitoring, simulation, and data-driven decision-making, addressing the multifaceted challenges of modern water distribution systems.

In the second part of this thesis, we shift focus to security, particularly IoT authentication and ML security.

We present our novel framework, RIOT-AKA, demonstrating that well-established au-

thentication mechanisms, long used in the cellular networking domain, can be adapted effectively for IoT environments. These mechanisms, known for their simplicity and robustness in handling roaming models, are well-suited to the dynamic nature of IoT deployments, where devices frequently transition between smart spaces. By substituting specialized cryptographic algorithms, we achieve a practical and accessible authentication solution tailored for IoT devices.

We finally explore ML security with a focus on adversarial machine learning (AML). Our key contribution is the development of a Subgraph Extraction-based Attack (SSEA), which isolates sections of CNNs that respond to semantically similar classes. SSEA reduces computational cost, achieving up to 20.66% reduction in FLOPs on CIFAR-100 and 9.57% on Tiny-Imagenet, with parameter reductions reaching up to 43.82% and 21%, respectively. Importantly, these subgraphs retain high transferability and often outperform the original CNNs, offering valuable insights into CNNs vulnerabilities and practical solutions for addressing AML challenges.

Bibliography

[1] Laith Farhan et al. “Energy efficiency for green internet of things (IoT) networks: A survey”. In: *Network* 1.3 (2021), pp. 279–314.

[2] William Stallings. *Foundations of modern networking: SDN, NFV, QoE, IoT, and Cloud*. Addison-Wesley Professional, 2015.

[3] Vinay Kumar, Sujay Mohan, and Rakesh Kumar. “A voice based one step solution for bulk IoT device onboarding”. In: *2019 16th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE. 2019, pp. 1–6.

[4] Lu Tan and Neng Wang. “Future internet: The internet of things”. In: *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*. Vol. 5. IEEE. 2010, pp. V5–376.

[5] Kazhan Othman Mohammed Salih et al. “A comprehensive survey on the Internet of Things with the industrial marketplace”. In: *Sensors* 22.3 (2022), p. 730.

[6] Yuan Li et al. “Towards a theoretical framework of strategic decision, supporting capability and information sharing under the context of Internet of Things”. In: *Information Technology and Management* 13 (2012), pp. 205–216.

[7] Mohammad Aazam, Sherali Zeadally, and Khaled A Harras. “Deploying fog computing in industrial internet of things and industry 4.0”. In: *IEEE Transactions on Industrial Informatics* 14.10 (2018), pp. 4674–4682.

[8] Usman Ahmad et al. “Survey on internet of things (iot) for different industry environments”. In: *Annals of Emerging Technologies in Computing (AETiC), Print ISSN* (2019), pp. 2516–0281.

[9] Shancang Li, Li Da Xu, and Xinheng Wang. “Compressed sensing signal and data acquisition in wireless sensor networks and internet of things”. In: *IEEE transactions on industrial informatics* 9.4 (2012), pp. 2177–2186.

[10] Wu He and Li Da Xu. “Integration of distributed enterprise applications: A survey”. In: *IEEE Transactions on industrial informatics* 10.1 (2012), pp. 35–42.

[11] Domenico Garlisi et al. “Leakage detection via edge processing in LoRaWAN-based smart water distribution networks”. In: *2022 18th International Conference on Mobility, Sensing and Networking (MSN)*. IEEE. 2022, pp. 223–230.

[12] Usman Raza, Parag Kulkarni, and Mahesh Sooriyabandara. “Low power wide area networks: An overview”. In: *ieee communications surveys & tutorials* 19.2 (2017), pp. 855–873.

[13] Warish D Patel et al. “NXTGeUH: LoRaWAN based NEXT generation ubiquitous healthcare system for vital signs monitoring & falls detection”. In: *2018 IEEE Punecon*. IEEE. 2018, pp. 1–8.

[14] Shwetank Dattatraya Mamdiwar et al. “Recent advances on IoT-assisted wearable sensor systems for healthcare monitoring”. In: *Biosensors* 11.10 (2021), p. 372.

[15] Marco Centenaro et al. “Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios”. In: *IEEE Wireless Communications* 23.5 (2016), pp. 60–67.

[16] Zach Shelby, Klaus Hartke, and Carsten Bormann. *The constrained application protocol (CoAP)*. Tech. rep. 2014.

[17] Danish Bilal Ansari, Atteeq-Ur Rehman, and Rizwan Ali. “Internet of things (iot) protocols: a brief exploration of mqtt and coap”. In: *International Journal of Computer Applications* 179.27 (2018), pp. 9–14.

[18] P Sommer et al. “Message-oriented middleware for industrial production systems”. In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. IEEE. 2018, pp. 1217–1223.

[19] Manel Houimli, Laid Kahloul, and Sihem Benaoun. “Formal specification, verification and evaluation of the MQTT protocol in the Internet of Things”. In: *2017 International conference on mathematics and information technology (ICMIT)*. IEEE. 2017, pp. 214–221.

[20] Zhenghe Wang et al. “Kafka and its using in high-throughput and reliable message distribution”. In: *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*. IEEE. 2015, pp. 117–120.

[21] Jay Kreps, Neha Narkhede, Jun Rao, et al. “Kafka: A distributed messaging system for log processing”. In: *Proceedings of the NetDB*. Vol. 11. 2011. Athens, Greece. 2011, pp. 1–7.

[22] Asif Ali Laghari et al. “A review and state of art of Internet of Things (IoT)”. In: *Archives of Computational Methods in Engineering* (2021), pp. 1–19.

[23] Daniel Minoli and Benedict Occhiogrosso. “Practical aspects for the integration of 5G networks and IoT applications in smart cities environments”. In: *Wireless Communications and Mobile Computing* 2019.1 (2019), p. 5710834.

[24] Debasish Mondal. “The internet of thing (IoT) and industrial automation: A future perspective”. In: *World J. Model. Simul* 15.2 (2019), pp. 140–149.

[25] Li Da Xu, Wu He, and Shancang Li. “Internet of things in industries: A survey”. In: *IEEE Transactions on industrial informatics* 10.4 (2014), pp. 2233–2243.

[26] Marco Lombardi, Francesco Pascale, and Domenico Santaniello. “Internet of things: A general overview between architectures, protocols and applications”. In: *Information* 12.2 (2021), p. 87.

[27] Mohammed Riyadh Abdmeziem, Djamel Tandjaoui, and Imed Romdhani. “Architecting the internet of things: state of the art”. In: *Robots and Sensor Clouds* (2016), pp. 55–75.

[28] Ala Al-Fuqaha et al. “Internet of things: A survey on enabling technologies, protocols, and applications”. In: *IEEE communications surveys & tutorials* 17.4 (2015), pp. 2347–2376.

[29] Henry Muccini and Mahyar Tourchi Moghaddam. “IoT architectural styles: A systematic mapping study”. In: *Software Architecture: 12th European Conference on Software Architecture, ECSA 2018, Madrid, Spain, September 24–28, 2018, Proceedings 12*. Springer. 2018, pp. 68–85.

[30] Dominique Guinard et al. “Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services”. In: *IEEE transactions on Services Computing* 3.3 (2010), pp. 223–235.

[31] Kiev Gama, Lionel Touseau, and Didier Donsez. “Combining heterogeneous service technologies for building an Internet of Things middleware”. In: *Computer Communications* 35.4 (2012), pp. 405–417.

[32] Mohammed H Alsharif et al. “Green IoT: A review and future research directions”. In: *Symmetry* 15.3 (2023), p. 757.

[33] Muhammad Adil et al. “An efficient load balancing scheme of energy gauge nodes to maximize the lifespan of constraint oriented networks”. In: *IEEE Access* 8 (2020), pp. 148510–148527.

[34] Jasenka Dizdarević et al. “A survey of communication protocols for internet of things and related challenges of fog and cloud computing integration”. In: *ACM Computing Surveys (CSUR)* 51.6 (2019), pp. 1–29.

[35] VC Shabna, K Jamshid, and S Manoj Kumar. “Energy minimization by removing data redundancy in wireless sensor networks”. In: *2014 International Conference on Communication and Signal Processing*. IEEE. 2014, pp. 1658–1663.

[36] Muhammad Adil et al. “MAC-AODV based mutual authentication scheme for constraint oriented networks”. In: *Ieee Access* 8 (2020), pp. 44459–44469.

[37] Erwin Adi et al. “Machine learning and data analytics for the IoT”. In: *Neural computing and applications* 32 (2020), pp. 16205–16233.

[38] Shree Krishna Sharma and Xianbin Wang. “Live data analytics with collaborative edge and cloud processing in wireless IoT networks”. In: *IEEE Access* 5 (2017), pp. 4621–4635.

[39] Shikhar Verma et al. “A survey on network methodologies for real-time analytics of massive IoT data and open research issues”. In: *IEEE Communications Surveys & Tutorials* 19.3 (2017), pp. 1457–1477.

[40] Weisong Shi et al. “Edge computing: Vision and challenges”. In: *IEEE internet of things journal* 3.5 (2016), pp. 637–646.

[41] Hung Cao et al. “Analytics everywhere: generating insights from the internet of things”. In: *Ieee Access* 7 (2019), pp. 71749–71769.

[42] Flavio Bonomi et al. “Fog computing and its role in the internet of things”. In: *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. 2012, pp. 13–16.

[43] Fog Computing. “the Internet of Things: Extend the Cloud to Where the Things are”. In: *Cisco White Paper* 13 (2015).

[44] Santosh Gautam Kashid and Sanjay A Pardeshi. “A survey of water distribution system and new approach to intelligent water distribution system”. In: *2014 First International Conference on Networks & Soft Computing (ICNSC2014)*. IEEE. 2014, pp. 339–344.

[45] Hajar Maseeh Yasin et al. “IoT and ICT based smart water management, monitoring and controlling system: A review”. In: *Asian Journal of Research in Computer Science* 8.2 (2021), pp. 42–56.

[46] Aurora González-Vidal, Jesús Cuenca-Jara, and Antonio F Skarmeta. “IoT for water management: Towards intelligent anomaly detection”. In: *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*. IEEE. 2019, pp. 858–863.

[47] Melike Erol-Kantarci and Hussein T Mouftah. “Wireless sensor networks for cost-efficient residential energy management in the smart grid”. In: *IEEE Transactions on Smart Grid* 2.2 (2011), pp. 314–325.

[48] Lakshmi Kanthan Narayanan and Suresh Sankaranarayanan. “IoT enabled smart water distribution and underground pipe health monitoring architecture for smart cities”. In: *2019 IEEE 5th international conference for convergence in technology (I2CT)*. IEEE. 2019, pp. 1–7.

[49] Z Ageed et al. “Cloud computing resources impacts on heavy-load parallel processing approaches”. In: *IOSR Journal of Computer Engineering (IOSR-JCE)* 22.3 (2020), pp. 30–41.

[50] Xiong Xiong et al. “Low power wide area machine-to-machine networks: Key techniques and prototype”. In: *IEEE Communications Magazine* 53.9 (2015), pp. 64–71.

[51] Lewis A Rossman et al. “EPANET 2: users manual”. In: (2000).

[52] Necati Ağıralioğlu. “Comparison of Friction Losses in Long Polyethylene Pipe Systems Using”. In: ().

[53] Katherine A Klise, Regan Murray, and Terra Haxton. “An overview of the water network tool for resilience (WNTR).” In: (2018).

[54] U.S. Environmental Protection Agency. *Water Network Tool for Resilience (WNTR)*. `https://usepa.github.io/WNTR/index.html`. Accessed: September 2024.

[55] Iqbal H Sarker. “Machine learning: Algorithms, real-world applications and research directions”. In: *SN computer science* 2.3 (2021), p. 160.

[56] Iqbal H Sarker, Md Hasan Furhad, and Raza Nowrozy. “Ai-driven cybersecurity: an overview, security intelligence modeling and research directions”. In: *SN Computer Science* 2.3 (2021), p. 173.

[57] European Commission. *Industry 5.0 - Towards a Sustainable, Human-Centric, and Resilient European Industry*. Accessed: 2024-10-10. 2023. url: `https://research-and-innovation.ec.europa.eu/research-area/industrial-research-and-innovation/industry-50_en`.

[58] Iqbal H Sarker et al. “Mobile data science and intelligent apps: concepts, AI-based modeling and research directions”. In: *Mobile Networks and Applications* 26.1 (2021), pp. 285–303.

[59] Iqbal H Sarker et al. “Cybersecurity data science: an overview from machine learning perspective”. In: *Journal of Big data* 7 (2020), pp. 1–29.

[60] Beata Ślusarczyk. “Industry 4.0–are we ready?” In: *Polish Journal of Management Studies* 17.1 (2018), pp. 232–248.

[61] Mahbod Tavallaee et al. “A detailed analysis of the KDD CUP 99 data set”. In: *2009 IEEE symposium on computational intelligence for security and defense applications*. Ieee. 2009, pp. 1–6.

[62] Nour Moustafa and Jill Slay. “UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)”. In: *2015 military communications and information systems conference (MilCIS)*. IEEE. 2015, pp. 1–6.

[63] Santi Phithakkitnukoon et al. “Behavior-based adaptive call predictor”. In: *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 6.3 (2011), pp. 1–28.

[64] Iqbal H Sarker et al. “Phone call log as a context source to modeling individual user behavior”. In: *Proceedings of the 2016 ACM international joint conference on pervasive and ubiquitous computing: adjunct*. 2016, pp. 630–634.

[65] Nathan Eagle and Alex Pentland. “Reality mining: sensing complex social systems”. In: *Personal and ubiquitous computing* 10 (2006), pp. 255–268.

[66] Fabrizio Balducci, Donato Impedovo, and Giuseppe Pirlo. “Machine learning applications on agricultural datasets for smart farm enhancement”. In: *Machines* 6.3 (2018), p. 38.

[67] Vijay Khadse, Parikshit N Mahalle, and Swapnil V Biraris. “An empirical comparison of supervised machine learning algorithms for internet of things data”. In: *2018 fourth international conference on computing communication control and automation (ICCUBEA)*. IEEE. 2018, pp. 1–6.

[68] Prasanth Lade, Rumi Ghosh, and Soundar Srinivasan. “Manufacturing analytics and industrial internet of things”. In: *IEEE Intelligent Systems* 32.3 (2017), pp. 74–79.

[69] Saima Safdar et al. “Machine learning based decision support systems (DSS) for heart disease diagnosis: a review”. In: *Artificial Intelligence Review* 50.4 (2018), pp. 597–623.

[70] Sajida Perveen et al. “Metabolic syndrome and development of diabetes mellitus: predictive modeling based on machine learning techniques”. In: *IEEE Access* 7 (2018), pp. 1365–1375.

[71] Tao Zheng et al. “A machine learning-based framework to identify type 2 diabetes through electronic health records”. In: *International journal of medical informatics* 97 (2017), pp. 120–127.

[72] Stephanie A Harmon et al. “Artificial intelligence for the detection of COVID-19 pneumonia on chest CT using multinational datasets”. In: *Nature communications* 11.1 (2020), p. 4080.

[73] Youssoufa Mohamadou, Aminou Halidou, and Pascalin Tiam Kapen. “A review of mathematical modeling, artificial intelligence and datasets used in the study, prediction and management of COVID-19”. In: *Applied Intelligence* 50.11 (2020), pp. 3913–3925.

[74] Mohssen Mohammed, Muhammad Badruddin Khan, and Eihab Bashier Mohammed Bashier. *Machine learning: algorithms and applications*. Crc Press, 2016.

[75] Jiawei Han, Jian Pei, and Hanghang Tong. *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

[76] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[77] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. “Reinforcement learning: A survey”. In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[78] David Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362.6419 (2018), pp. 1140–1144.

[79] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *the Journal of machine Learning research* 12 (2011), pp. 2825–2830.

[80] George H John and Pat Langley. “Estimating continuous distributions in Bayesian classifiers”. In: *arXiv preprint arXiv:1302.4964* (2013).

[81] S. Sathiya Keerthi et al. “Improvements to Platt’s SMO algorithm for SVM classifier design”. In: *Neural computation* 13.3 (2001), pp. 637–649.

[82] J. Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1 (1986), pp. 81–106.

[83] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[84] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[85] Iqbal H Sarker, ASM Kayes, and Paul Watters. “Effectiveness analysis of machine learning classification models for predicting personalized context-aware smartphone usage”. In: *Journal of Big Data* 6.1 (2019), pp. 1–28.

[86] Longbing Cao. “Data science: a comprehensive overview”. In: *ACM Computing Surveys (CSUR)* 50.3 (2017), pp. 1–42.

[87] Mohammad Saeid Mahdavinejad et al. “Machine learning for Internet of Things data analysis: A survey”. In: *Digital Communications and Networks* 4.3 (2018), pp. 161–175.

[88] Ian H Witten and Eibe Frank. “Data mining: practical machine learning tools and techniques with Java implementations”. In: *Acm Sigmod Record* 31.1 (2002), pp. 76–77.

[89] Juan Guerrero-Ibáñez, Sherali Zeadally, and Juan Contreras-Castillo. “Sensor technologies for intelligent transportation systems”. In: *Sensors* 18.4 (2018), p. 1212.

[90] Azzedine Boukerche and Jiahao Wang. “Machine learning-based traffic prediction models for intelligent transportation systems”. In: *Computer Networks* 181 (2020), p. 107530.

[91] Aniekan Essien et al. “Improving urban traffic speed prediction using data source fusion and deep learning”. In: *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE. 2019, pp. 1–8.

[92] Aniekan Essien et al. “A deep-learning model for urban traffic flow prediction with traffic events mined from twitter”. In: *World Wide Web* 24.4 (2021), pp. 1345–1368.

[93] Hironobu Fujiyoshi, Tsubasa Hirakawa, and Takayoshi Yamashita. “Deep learning-based image recognition for autonomous driving”. In: *IATSS research* 43.4 (2019), pp. 244–252.

[94] Rohit Sharma et al. “A systematic literature review on machine learning applications for sustainable agriculture supply chain performance”. In: *Computers & Operations Research* 119 (2020), p. 104926.

[95] Nadia Adnan et al. “The effects of knowledge transfer on farmers decision making toward sustainable agriculture practices: In view of green fertilizer technology”. In: *World Journal of Science, Technology and Sustainable Development* 15.1 (2018), pp. 98–115.

[96] Sachin S Kamble, Angappa Gunasekaran, and Shradha A Gawankar. “Sustainable Industry 4.0 framework: A systematic literature review identifying the current trends and future perspectives”. In: *Process safety and environmental protection* 117 (2018), pp. 408–425.

[97] Sachin S Kamble, Angappa Gunasekaran, and Shradha A Gawankar. “Achieving sustainable performance in a data-driven agriculture supply chain: A review for research and applications”. In: *International Journal of Production Economics* 219 (2020), pp. 179–194.

[98] Ian Goodfellow. *Deep learning*. 2016.

[99] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *nature* 521.7553 (2015), pp. 436–444.

[100] Yang Xin et al. “Machine learning and deep learning methods for cybersecurity”. In: *Ieee access* 6 (2018), pp. 35365–35381.

[101] Ying Zhang and Chen Ling. “A strategy to apply machine learning to small datasets in materials science”. In: *Npj Computational Materials* 4.1 (2018), p. 25.

[102] Cynthia Rudin. “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead”. In: *Nature machine intelligence* 1.5 (2019), pp. 206–215.

[103] John Searle. *Minds, Brains, and Programs*. 1980.

[104] Jiuxiang Gu et al. “Recent advances in convolutional neural networks”. In: *Pattern recognition* 77 (2018), pp. 354–377.

[105] Shi Dong, Ping Wang, and Khushnood Abbas. “A survey on deep learning and its applications”. In: *Computer Science Review* 40 (2021), p. 100379.

[106] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems* 25 (2012).

[107] Michael M Bronstein et al. “Geometric deep learning: going beyond euclidean data”. In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.

[108] Zonghan Wu et al. “A comprehensive survey on graph neural networks”. In: *IEEE transactions on neural networks and learning systems* 32.1 (2020), pp. 4–24.

[109] Alessandro Sperduti and Antonina Starita. “Supervised neural networks for the classification of structures”. In: *IEEE transactions on neural networks* 8.3 (1997), pp. 714–735.

[110] Marco Gori, Gabriele Monfardini, and Franco Scarselli. “A new model for learning in graph domains”. In: *Proceedings. 2005 IEEE international joint conference on neural networks, 2005.* Vol. 2. IEEE. 2005, pp. 729–734.

[111] Franco Scarselli et al. “The graph neural network model”. In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[112] Claudio Gallicchio and Alessio Micheli. “Graph echo state networks”. In: *The 2010 international joint conference on neural networks (IJCNN)*. IEEE. 2010, pp. 1–8.

[113] Yujia Li et al. “Gated graph sequence neural networks”. In: *arXiv preprint arXiv:1511.05493* (2015).

[114] Hanjun Dai et al. “Learning steady-states of iterative algorithms over graphs”. In: *International conference on machine learning*. PMLR. 2018, pp. 1106–1114.

[115] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).

[116] Muhan Zhang et al. “An end-to-end deep learning architecture for graph classification”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[117] Zhitao Ying et al. “Hierarchical graph representation learning with differentiable pooling”. In: *Advances in neural information processing systems* 31 (2018).

[118] Shirui Pan et al. “Joint structure feature exploration and regularization for multi-task graph classification”. In: *IEEE Transactions on Knowledge and Data Engineering* 28.3 (2015), pp. 715–728.

[119] Shirui Pan et al. “Task sensitive feature exploration and learning for multitask graph classification”. In: *IEEE transactions on cybernetics* 47.3 (2016), pp. 744–758.

[120] Thomas N Kipf and Max Welling. “Variational graph auto-encoders”. In: *arXiv preprint arXiv:1611.07308* (2016).

[121] Shirui Pan et al. “Adversarially regularized graph autoencoder for graph embedding”. In: *arXiv preprint arXiv:1802.04407* (2018).

[122] Will Hamilton, Zhitao Ying, and Jure Leskovec. “Inductive representation learning on large graphs”. In: *Advances in neural information processing systems* 30 (2017).

[123] Joan Bruna et al. “Spectral networks and locally connected networks on graphs”. In: *arXiv preprint arXiv:1312.6203* (2013).

[124] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. “Convolutional neural networks on graphs with fast localized spectral filtering”. In: *Advances in neural information processing systems* 29 (2016).

[125] Alessio Micheli. “Neural network for graphs: A contextual constructive approach”. In: *IEEE Transactions on Neural Networks* 20.3 (2009), pp. 498–511.

[126] Justin Gilmer et al. “Neural message passing for quantum chemistry”. In: *International conference on machine learning*. PMLR. 2017, pp. 1263–1272.

[127] Ron Levie et al. “Cayleynets: Graph convolutional neural networks with complex rational spectral filters”. In: *IEEE Transactions on Signal Processing* 67.1 (2018), pp. 97–109.

[128] James Atwood and Don Towsley. “Diffusion-convolutional neural networks”. In: *Advances in neural information processing systems* 29 (2016).

[129] Yaguang Li et al. “Diffusion convolutional recurrent neural network: Data-driven traffic forecasting”. In: *arXiv preprint arXiv:1707.01926* (2017).

[130] Tyler Derr, Yao Ma, and Jiliang Tang. “Signed graph convolutional networks”. In: *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE. 2018, pp. 929–934.

[131] Felipe Petroski Such et al. “Robust spatial filtering with graph convolutional neural networks”. In: *IEEE Journal of Selected Topics in Signal Processing* 11.6 (2017), pp. 884–896.

[132] Xiao Wang et al. “Heterogeneous graph attention network”. In: *The world wide web conference*. 2019, pp. 2022–2032.

[133] Matthias Fey and Jan Eric Lenssen. *GINEConv Layer in PyTorch Geometric*. `https://pytorch-geometric.readthedocs.io/en/latest/generated/torch_geometric.nn.conv.GINEConv.html`. Accessed: 2024-10-05. 2019.

[134] Weihua Hu et al. “Strategies for pre-training graph neural networks”. In: *arXiv preprint arXiv:1905.12265* (2019).

[135] Guofu Li et al. “Security matters: A survey on adversarial machine learning”. In: *arXiv preprint arXiv:1810.07339* (2018).

[136] Kathrin Grosse et al. “Adversarial perturbations against deep neural networks for malware classification”. In: *arXiv preprint arXiv:1606.04435* (2016).

[137] George E Dahl et al. “Large-scale malware classification using random projections and neural networks”. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 3422–3426.

[138] Guy Katz et al. “Reluplex: An efficient SMT solver for verifying deep neural networks”. In: *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer. 2017, pp. 97–117.

[139] Kexin Pei et al. “Deepxplore: Automated whitebox testing of deep learning systems”. In: *proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 1–18.

[140] Ling Huang et al. “Adversarial machine learning”. In: *Proceedings of the 4th ACM workshop on Security and artificial intelligence*. 2011, pp. 43–58.

[141] C Szegedy. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013).

[142] Marco Barreno et al. “Can machine learning be secure?” In: *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. 2006, pp. 16–25.

[143] Nicolas Papernot et al. “Practical black-box attacks against machine learning”. In: *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 2017, pp. 506–519.

[144] Nina Narodytska and Shiva Prasad Kasiviswanathan. “Simple Black-Box Adversarial Attacks on Deep Neural Networks.” In: *CVPR Workshops*. Vol. 2. 2. 2017.

[145] Seyed-Mohsen Moosavi-Dezfooli et al. “Universal adversarial perturbations”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1765–1773.

[146] U. Wwap. “World Water Assessment Programme: The United Nations World Water Development Report 4: Managing Water under Uncertainty and Risk”. In: (2012), p. 39.

[147] *Europe’s Water in Figures. 2021 Edition*. ISBN 978-2-9602226-3-0. EurEau, 2021. url: https://www.eureau.org/resources/publications/eureau-publications/5824-europe-s-water-in-figures-2021/file.

[148] Victor K Sarker et al. “A survey on LoRa for IoT: Integrating edge computing”. In: *2019 Fourth International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE. 2019, pp. 295–300.

[149] Dimitrios Amaxilatis et al. “A smart water metering deployment based on the fog computing paradigm”. In: *Applied Sciences* 10.6 (2020), p. 1965.

[150] IoT Analytics. *LPWAN Market Report 2018-2023*. Accessed: 31/08/2022. Sept. 2018. url: https://iot-analytics.com/product/lpwan-market-report-2018-2023/.

[151] Yahui Meng et al. “Advancing the state of the fog computing to enable 5g network technologies”. In: *Sensors* 20.6 (2020), p. 1754.

[152] Ahsan Rafiq et al. “Intelligent edge computing enabled reliable emergency data transmission and energy efficient offloading in 6TiSCH-based IIoT networks”. In: *Sustainable Energy Technologies and Assessments* 53 (2022), p. 102492.

[153] Mohammed Rezwanul Islam et al. “A review on current technologies and future direction of water leakage detection in water distribution network”. In: *IEEE Access* 10 (2022), pp. 107177–107201.

[154] OpenWaterAnalytics. *EPANET Example Network Model*. Accessed: 2022-08-28. n.d. url: https://raw.githubusercontent.com/OpenWaterAnalytics/epanet-example-networks/master/epanet-tests/large/NW_Model.inp.

[155] LoRaSim Team. *LoRaSim: Discrete-Event Simulator in SimPy for Simulating Collisions in LoRa Networks*. Accessed: 2020-04-16. n.d. url: http://www.lancaster.ac.uk/scc/sites/lora/.

[156] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. "The Google File System". In: *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*. SOSP '03. Bolton Landing, NY, USA: Association for Computing Machinery, 2003, pp. 29–43. isbn: 1581137575. doi: 10.1145/945445.945450. url: https://doi.org/10.1145/945445.945450.

[157] Matei Zaharia et al. "Spark: Cluster Computing with Working Sets". In: *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. HotCloud'10. Boston, MA, 2010, p. 10.

[158] Alexander Alexandrov and et al. "The Stratosphere Platform for Big Data Analytics". In: *The VLDB Journal* 23.6 (Dec. 2014), pp. 939–964. issn: 1066-8888. doi: 10.1007/s00778-014-0357-y. url: https://doi.org/10.1007/s00778-014-0357-y.

[159] Steffen Zeuch et al. "The nebulastream platform: Data and application management for the internet of things". In: *arXiv preprint arXiv:1910.07867* (2019).

[160] Domenico Garlisi et al. "Real-Time Leakage Zone Detection in Water Distribution Networks: A Machine Learning-Based Stream Processing Algorithm". In: *International Symposium on Algorithmic Aspects of Cloud Computing*. Springer. 2023, pp. 86–99.

[161] Terry Randall and R Koech. "Smart water metering technology for water management in urban areas". In: *Water eJ* 4 (2019), pp. 1–14.

[162] Adrià Soldevila et al. "Leak localization in water distribution networks using a mixed model-based/data-driven approach". In: *Control Engineering Practice* 55 (2016), pp. 162–173.

[163] Adrià Soldevila et al. "Leak localization in water distribution networks using Bayesian classifiers". In: *Journal of Process Control* 55 (2017), pp. 1–9.

[164] Jetmir Haxhibeqiri et al. "A survey of LoRaWAN for IoT: From technology to application". In: *Sensors* 18.11 (2018), p. 3995.

[165] Nagib Matni et al. "LoRaWAN gateway placement model for dynamic Internet of Things scenarios". In: *Sensors* 20.15 (2020), p. 4336.

[166] Qahhar Muhammad Qadir et al. "Low power wide area networks: A survey of enabling technologies, applications and interoperability needs". In: *IEEE Access* 6 (2018), pp. 77454–77473.

[167] Tara Petrić et al. “Measurements, performance and analysis of LoRa FABIAN, a real-world implementation of LPWAN”. In: *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. IEEE. 2016, pp. 1–7.

[168] Andrew J Wixted et al. “Evaluation of LoRa and LoRaWAN for wireless sensor networks”. In: *2016 IEEE SENSORS*. IEEE. 2016, pp. 1–3.

[169] Ramon Sanchez-Iborra et al. “Performance evaluation of LoRa considering scenario conditions”. In: *Sensors* 18.3 (2018), p. 772.

[170] Hongxian Tian, Mary Ann Weitnauer, and Gedeon Nyengele. “Optimized gateway placement for interference cancellation in transmit-only LPWA networks”. In: *Sensors* 18.11 (2018), p. 3884.

[171] Huy Truong et al. “Graph neural networks for pressure estimation in water distribution systems”. In: *Water Resources Research* 60.7 (2024), e2023WR036741.

[172] Jorge E Pesantez et al. “Using a digital twin to explore water infrastructure impacts during the COVID-19 pandemic”. In: *Sustainable Cities and Society* 77 (2022), p. 103520.

[173] Fabrício José Pontes et al. “Design of experiments and focused grid search for neural network parameter optimization”. In: *Neurocomputing* 186 (2016), pp. 22–34.

[174] Y Hamam and KS Hindi. “Optimised on-line leakage minimisation in water piping networks using neural nets”. In: *Proceedings of the IFIP Working Conference, Dagschul, Germany*. Vol. 28. 1992, pp. 57–64.

[175] Pierre Carpentier and Guy Cohen. “Applied mathematics in water supply network management”. In: *Automatica* 29.5 (1993), pp. 1215–1250.

[176] Gabriela Cembrano et al. “Optimal control of a water distribution network in a supervisory control system”. In: *Control engineering practice* 8.10 (2000), pp. 1177–1188.

[177] Oladipupo Bello et al. “Solving management problems in water distribution networks: A survey of approaches and mathematical models”. In: *Water* 11.3 (2019), p. 562.

[178] Michael Grieves and John Vickers. “Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems”. In: *Transdisciplinary perspectives on complex systems: New findings and approaches* (2017), pp. 85–113.

[179] Yiwen Wu, Ke Zhang, and Yan Zhang. “Digital twin networks: A survey”. In: *IEEE Internet of Things Journal* 8.18 (2021), pp. 13789–13804.

[180] Helena M Ramos et al. “Smart water grids and digital twin for the management of system efficiency in water distribution networks”. In: *Water* 15.6 (2023), p. 1129.

[181] Modelica Association et al. “Modelica-a unified object-oriented language for systems modeling”. In: *Language Specification, Version* 3.04 (2017).

[182] Michael Schluse et al. “Experimentable digital twins—Streamlining simulation-based systems engineering for industry 4.0”. In: *IEEE Transactions on industrial informatics* 14.4 (2018), pp. 1722–1731.

[183] Didem Gürdür Broo et al. “Built environment of Britain in 2040: Scenarios and strategies”. In: *Sustainable cities and society* 65 (2021), p. 102645.

[184] Fei Tao et al. “Digital twin in industry: State-of-the-art”. In: *IEEE Transactions on industrial informatics* 15.4 (2018), pp. 2405–2415.

[185] Ragunathan Rajkumar et al. “Cyber-physical systems: the next computing revolution”. In: *Proceedings of the 47th design automation conference*. 2010, pp. 731–736.

[186] Francesco Gino Ciliberti et al. “From digital twin paradigm to digital water services”. In: *Journal of Hydroinformatics* 25.6 (2023), pp. 2444–2459.

[187] Carlos A Bonilla et al. “A digital twin of a water distribution system by using graph convolutional networks for pump speed-based state estimation”. In: *Water* 14.4 (2022), p. 514.

[188] Nikolaj T Mücke et al. “A probabilistic digital twin for leak localization in water distribution networks using generative deep learning”. In: *Sensors* 23.13 (2023), p. 6179.

[189] Miguel Grinberg. *Flask web development*. ” O’Reilly Media, Inc.”, 2018.

[190] Elijah Meeks. *D3. js in Action: Data visualization with JavaScript*. Simon and Schuster, 2017.

[191] Eclipse Foundation. *Eclipse Ditto*. `https://eclipse.dev/ditto/`. Accessed: 2024-10-05.

[192] Narges Yousefnezhad, Avleen Malhi, and Kary Främling. “Security in product lifecycle of IoT devices: A survey”. In: *Journal of Network and Computer Applications* 171 (2020), p. 102779.

[193] Omar Abdel Wahab et al. “Optimal load distribution for the detection of VM-based DDoS attacks in the cloud”. In: *IEEE transactions on services computing* 13.1 (2017), pp. 114–129.

[194] Antonio L Maia Neto et al. “Aot: Authentication and access control for the entire iot device life-cycle”. In: *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM*. 2016, pp. 1–15.

[195] Jun Young Kim et al. “ESIoT: Enabling secure management of the internet of things”. In: *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*. 2017, pp. 219–229.

[196] Georgios Selimis et al. “Rescure: A security solution for iot life cycle”. In: *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 2020, pp. 1–10.

[197] David Kotz and Travis Peters. “Challenges to ensuring human safety throughout the life-cycle of Smart Environments”. In: *Proceedings of the 1st ACM Workshop on the Internet of Safe Things*. 2017, pp. 1–7.

[198] Yu Zheng et al. “AKA and authorization scheme for 4G mobile networks based on trusted mobile platform”. In: *2005 5th International Conference on Information Communications & Signal Processing*. IEEE. 2005, pp. 976–980.

[199] Emmanuel Baccelli et al. “RIOT OS: Towards an OS for the Internet of Things”. In: *2013 IEEE conference on computer communications workshops (INFOCOM WKSHPS)*. IEEE. 2013, pp. 79–80.

[200] Emmanuel Baccelli et al. “RIOT: An open source operating system for low-end embedded devices in the IoT”. In: *IEEE Internet of Things Journal* 5.6 (2018), pp. 4428–4440.

[201] Stephanie Alt et al. “A cryptographic analysis of umts/lte aka”. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2016, pp. 18–35.

[202] Giuseppe Bianchi, Alberto La Rosa, and Gabriele Restuccia. “RIOT-AKA: cellular-like authentication over IoT devices”. In: *2021 IEEE 29th International Conference on Network Protocols (ICNP)*. IEEE. 2021, pp. 1–6.

[203] RIOT OS. *RIOT: The friendly Operating System for the Internet of Things*. `https://riot-os.org/`. Accessed: September 2024.

[204] RIOT OS GitHub. *RIOT OS Wiki on GitHub*. `https://github.com/RIOT-OS/RIOT/wiki`. Accessed: September 2024.

[205] Vangelis Gazis et al. “Generic system architecture for 4G mobile communications”. In: *The 57th IEEE Semiannual Vehicular Technology Conference, 2003. VTC 2003-Spring.* Vol. 3. IEEE. 2003, pp. 1512–1516.

[206] Hugo Krawczyk. “Cryptographic extraction and key derivation: The HKDF scheme”. In: *Annual Cryptology Conference*. Springer. 2010, pp. 631–648.

[207] Eric Rescorla. *The transport layer security (TLS) protocol version 1.3*. Tech. rep. 2018.

[208] Gabriele Restuccia, Hannes Tschofenig, and Emmanuel Baccelli. “Low-power IoT communication security: On the performance of DTLS and TLS 1.3”. In: *2020 9th IFIP International Conference on Performance Evaluation and Modeling in Wireless Networks (PEMWN)*. IEEE. 2020, pp. 1–6.

[209] G Selander et al. *RFC 8613: Object Security for Constrained RESTful Environments (OSCORE)*. 2019.

[210] A. La Rosa. *Deus-Ex-Mortis/RIOT-security*. *GitHub*. Available at: https://github.com/Deus-Ex-Mortis/RIOT-security/tree/RIOT-AKA (Accessed: September 2024).

[211] A. La Rosa. *Deus-Ex-Mortis/Aiocoap*. *GitHub*. Available at: https://github.com/Deus-Ex-Mortis/Aiocoap (Accessed: September 2024).

[212] H. Petersen. *haukepetersen/cosy: Python tool analyzing memory usage and distribution in .elf files*. *GitHub*. Available at: https://github.com/haukepetersen/cosy (Accessed: September 2024).

[213] Peter Kietzmann et al. "A PUF seed generator for riot: Introducing crypto-fundamentals to the wild". In: *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. 2018, pp. 513–513.

[214] Lieneke Kusters and Frans MJ Willems. "Secret-key capacity regions for multiple enrollments with an SRAM-PUF". In: *IEEE Transactions on Information Forensics and Security* 14.9 (2019), pp. 2276–2287.

[215] Lieneke Kusters and Frans MJ Willems. "Capacity of a dual enrollment system with two keys based on an SRAM-PUF". In: *2018 International Zurich Seminar on Information and Communication (IZS 2018)*. 2018, pp. 79–83.

[216] G Edward Suh and Srinivas Devadas. "Physical unclonable functions for device authentication and secret key generation". In: *Proceedings of the 44th annual design automation conference*. 2007, pp. 9–14.

[217] RIOT-OS. *SRAM PUF*. *RIOT-OS API Documentation*. Accessed: September 2024. url: https://api.riot-os.org/group__sys__puf__sram.html.

[218] Jeffery C Eidenshink. "Detection of leaks in buried rural water pipelines using thermal infrared images". In: *Photogrammetric Engineering and Remote Sensing* 51.5 (1985), pp. 561–564.

[219] Gary J Weil and Richard J Graf. "Infrared-thermography-based pipeline leak detection systems". In: *Thermosense XIII*. Vol. 1467. SPIE. 1991, pp. 18–33.

[220] Gary J Weil, Richard J Graf, and Leann M Forister. "Remote sensing pipeline rehabilitation methodologies based upon the utilization of infrared thermography". In: (1994).

[221] Ahmed Atef et al. "Multi-tier method using infrared photography and GPR to detect and locate water leaks". In: *Automation in Construction* 61 (2016), pp. 162–170.

[222] Nengsheng Bao et al. "A machine vision—Based pipe leakage detection system for automated power plant maintenance". In: *Sensors* 22.4 (2022), p. 1588.

[223] Renlian Zhou, Huaizhi Su, and Zhiping Wen. "Experimental study on leakage detection of grassed earth dam by passive infrared thermography". In: *NDT & E International* 126 (2022), p. 102583.

[224] Wallace WL Lai et al. “Perturbation mapping of water leak in buried water pipes via laboratory validation experiments with high-frequency ground penetrating radar (GPR)”. In: *Tunnelling and Underground Space Technology* 52 (2016), pp. 157–167.

[225] C Guo, K Shi, and X Chu. “Cross-correlation analysis of multiple fibre optic hydrophones for water pipeline leakage detection”. In: *International Journal of Environmental Science and Technology* (2022), pp. 1–12.

[226] Harshit Shukla and Kalyan Piratla. “Leakage detection in water pipelines using supervised classification of acceleration signals”. In: *Automation in Construction* 117 (2020), p. 103256.

[227] Jiheon Kang et al. “Novel leakage detection by ensemble CNN-SVM and graph-based localization in water distribution systems”. In: *IEEE Transactions on Industrial Electronics* 65.5 (2017), pp. 4279–4289.

[228] Weinan Xu et al. “Leakage identification in water pipes using explainable ensemble tree model of vibration signals”. In: *Measurement* 194 (2022), p. 110996.

[229] João Alves Coelho, André Glória, and Pedro Sebastião. “Precise water leak detection using machine learning and real-time sensor data”. In: *IoT* 1.2 (2020), pp. 474–493.

[230] Samer El-Zahab, Eslam Mohammed Abdelkader, and Tarek Zayed. “An accelerometer-based leak detection system”. In: *Mechanical Systems and Signal Processing* 108 (2018), pp. 276–291.

[231] Haofeng Li, Guanbin Li, and Yizhou Yu. “ROSA: Robust salient object detection against adversarial attacks”. In: *IEEE transactions on cybernetics* 50.11 (2019), pp. 4835–4847.

[232] Zhaohui Che et al. “Adversarial attack against deep saliency models powered by non-redundant priors”. In: *IEEE Transactions on Image Processing* 30 (2021), pp. 1973–1988.

[233] Yulong Wang et al. “Adversarial Attacks and Defenses in Machine Learning-Powered Networks: A Contemporary Survey”. In: *arXiv preprint arXiv:2303.06302* (2023).

[234] Hanan Hussain, PS Tamizharasan, and CS Rahul. “Design possibilities and challenges of DNN models: a review on the perspective of end devices”. In: *Artificial Intelligence Review* (2022), pp. 1–59.

[235] Ferheen Ayaz et al. “Improving Robustness Against Adversarial Attacks with Deeply Quantized Neural Networks”. In: *arXiv preprint arXiv:2304.12829* (2023).

[236] Nicholas Carlini and David Wagner. “Towards evaluating the robustness of neural networks”. In: *2017 ieee symposium on security and privacy (sp)*. Ieee. 2017, pp. 39–57.

[237] Zekun Zhang and Tianfu Wu. “Learning ordered top-k adversarial attacks via adversarial distillation”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*. 2020, pp. 776–777.

[238] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks”. In: *arXiv preprint arXiv:1706.06083* (2017).

[239] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).

[240] Harry24k. *adversarial-attacks-pytorch*. `https://github.com/Harry24k/adversarial-attacks-pytorch`. Accessed: Jan 2024. 2024.

[241] Ricardo Ribani and Mauricio Marengoni. “A survey of transfer learning for convolutional neural networks”. In: *2019 32nd SIBGRAPI conference on graphics, patterns and images tutorials (SIBGRAPI-T)*. IEEE. 2019, pp. 47–57.

[242] Yaqing Wang et al. “Generalizing from a few examples: A survey on few-shot learning”. In: *ACM computing surveys (csur)* 53.3 (2020), pp. 1–34.

[243] Jian Liang, Ran He, and Tieniu Tan. “A comprehensive survey on test-time adaptation under distribution shifts”. In: *International Journal of Computer Vision* (2024), pp. 1–34.

[244] Gaurav Menghani. “Efficient deep learning: A survey on making deep learning models smaller, faster, and better”. In: *ACM Computing Surveys* 55.12 (2023), pp. 1–37.

[245] Tala Talaei Khoei, Hadjar Ould Slimane, and Naima Kaabouch. “Deep learning: Systematic review, models, challenges, and research directions”. In: *Neural Computing and Applications* 35.31 (2023), pp. 23103–23124.

[246] Chao Li et al. “Adversarial attacks in computer vision: a survey”. In: *Journal of Membrane Computing* (2024), pp. 1–18.

[247] Sazzad Sayyed, Jonathan Ashdown, and Francesco Restuccia. “Faster and Accurate Neural Networks with Semantic Inference”. In: *arXiv preprint arXiv:2310.01259* (2023).

[248] Qian Huang et al. “Enhancing adversarial example transferability with an intermediate level attack”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 4733–4742.

[249] Nicolas Papernot et al. “Distillation as a defense to adversarial perturbations against deep neural networks”. In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 582–597.

[250] Nicolas Papernot et al. “The limitations of deep learning in adversarial settings”. In: *2016 IEEE European symposium on security and privacy (EuroS&P)*. IEEE. 2016, pp. 372–387.

[251] Pin-Yu Chen et al. “Ead: elastic-net attacks to deep neural networks via adversarial examples”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 32. 1. 2018.

[252] Yunxiao Qin et al. “Training meta-surrogate model for transferable adversarial attack”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 37. 8. 2023, pp. 9516–9524.

[253] Chuan Du et al. “Fast C&W: A fast adversarial attack algorithm to fool SAR target recognition with deep convolutional neural networks”. In: *IEEE Geoscience and Remote Sensing Letters* 19 (2021), pp. 1–5.

[254] Dongxian Wu et al. “Skip connections matter: On the transferability of adversarial examples generated with resnets”. In: *arXiv preprint arXiv:2002.05990* (2020).

[255] Zhewei Yao et al. “Trust region based adversarial attack on neural networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 11350–11359.

[256] Alexander Matyasko and Lap-Pui Chau. “Pdpgd: Primal-dual proximal gradient descent adversarial attack”. In: *arXiv preprint arXiv:2106.01538* (2021).

[257] Hanwei Zhang et al. “Walking on the edge: Fast, low-distortion adversarial examples”. In: *IEEE Transactions on Information Forensics and Security* 16 (2020), pp. 701–713.

[258] Vikash Sehwag et al. “Hydra: Pruning adversarially robust neural networks”. In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 19655–19666.

[259] Dongxian Wu and Yisen Wang. “Adversarial neuron pruning purifies backdoored deep models”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 16913–16925.

[260] Shaokai Ye et al. “Adversarial robustness vs. model compression, or both?” In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 111–120.

[261] Guneet S Dhillon et al. “Stochastic activation pruning for robust adversarial defense”. In: *arXiv preprint arXiv:1803.01442* (2018).

[262] Artur Jordao and Hélio Pedrini. “On the effect of pruning on adversarial robustness”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 1–11.

[263] Shaohui Lin et al. “Towards optimal structured cnn pruning via generative adversarial learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 2790–2799.

[264] Manoj-Rohit Vemparala et al. “Adversarial robust model compression using in-train pruning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 66–75.

[265] Karen Simonyan and Andrew Zisserman. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint arXiv:1409.1556* (2014).

[266] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[267] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[268] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. “A downsampled variant of imagenet as an alternative to the cifar datasets”. In: *arXiv preprint arXiv:1707.08819* (2017).

[269] Luis Balderas, Miguel Lastra, and José M Benítez. “Optimizing Convolutional Neural Network Architecture”. In: *arXiv preprint arXiv:2401.01361* (2023).

[270] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).

[271] Yang He and Lingao Xiao. “Structured pruning for deep convolutional neural networks: A survey”. In: *IEEE transactions on pattern analysis and machine intelligence* (2023).

[272] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems* 32 (2019).

[273] Hoki Kim. “Torchattacks: A pytorch repository for adversarial attacks”. In: *arXiv preprint arXiv:2010.01950* (2020).

[274] Nicolas Papernot et al. “Technical report on the cleverhans v2. 1.0 adversarial examples library”. In: *arXiv preprint arXiv:1610.00768* (2016).

[275] Gongfan Fang et al. “Depgraph: Towards any structural pruning”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 16091–16101.

[276] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. “Adversarial examples in the physical world”. In: *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018, pp. 99–112.

[277] Yinpeng Dong et al. “Boosting adversarial attacks with momentum”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 9185–9193.

[278] Leo Schwinn et al. “Exploring misclassifications of robust neural networks to enhance adversarial attacks”. In: *Applied Intelligence* (2023), pp. 1–17.

[279] Xuanqing Liu et al. “Adv-bnn: Improved adversarial defense through robust bayesian neural network”. In: *arXiv preprint arXiv:1810.01279* (2018).

[280] Jierun Chen et al. “Run, Don’t Walk: Chasing Higher FLOPS for Faster Neural Networks”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 12021–12031.