Predictive Fault Localization in Mobile Networks using Multi-Domain KPIs

Ryan Cui

Delft University of Technology

TU Delft

Predictive Fault Localization in Mobile Networks using Multi-Domain KPIs

by

Ryan Cui

Jian Yu Ryan Cui

to obtain the degree of Master of Science

Student number:	5114640
Supervisors:	Prof.dr.ir. R.E. Kooij
	Ing. M. Kloen
	Drs. M. Ouwens
Thesis Committee:	Dr. H. Wang
Publication date:	26^{th} of February, 2026
Faculty:	Faculty of Electrical Engineering,
	Mathematics and Computer Science, Delft

TU Delft

Abstract

The rapid growth of mobile data traffic and the evolution towards 5G-Advanced and 6G networks have significantly increased the operational complexity of mobile networks, making fault localization a critical challenge for mobile network operators. Traditional fault management approaches rely on reactive, threshold-based alarms operating within individual network domains. This is increasingly ineffective for detecting and localizing faults in complex, multi-domain environments.

This thesis proposes a cross-domain fault localization framework that combines unsupervised anomaly detection with offline reinforcement learning, where the RL agent learns a policy for identifying the most likely fault origin based on observed anomaly patterns across domains. The proposed framework analyzes time-series Key Performance Indicators (KPIs) collected from the Radio Access Network (RAN), Core Network and end-to-end (E2E) domains to detect anomalous behaviour without requiring labeled data. Subsequently, the detected anomalies and cross-domain results are used by an offline reinforcement learning agent to track the most probable origin of faults across network domains.

The framework is evaluated using real-world KPI data collected over a 1 month period, consisting of 13 KPIs across the 3 domains. Unsupervised anomaly detection is applied to identify deviations from normal network behavior, while fault localization is performed using reinforcement learning based on the observed anomaly patterns. The results demonstrate that the proposed approach can identify anomalies and provide fault localization despite the lack of explicit fault labels.

This thesis highlights key challenges such as traffic dependent KPI behaviour, noise during low traffic periods and threshold selection. While the results indicate that combining unsupervised anomaly detection with reinforcement learning is a promising direction for predictive fault localization, further refinement is required to improve robustness, precision and operational consistency. Future work should focus on online deployment, calibration and improved learning strategies to support deployment in real world mobile network environments.

Acknowledgements

Completing this thesis, conducted at TU Delft in collaboration with KPN has been a challenging yet rewarding journey and I would like to express my immense gratitude to the individuals and organizations who supported me throughout this process.

First and foremost, I extend my deepest appreciation to my supervisors: Prof.dr.ir. Rob Kooij of TU Delft, Ing. Monique Kloen and Drs. Marleen Ouwens from KPN, for their invaluable guidance, insightful feedback and continuous support. Furthermore, I want to thank Assistant Professor Edgar van Boven for facilitating the connection between KPN. Together, they provided a unique and enriching environment that was essential to the success of this thesis. I am especially grateful to the team TDO Network Mobile Access Network Optimization at KPN for offering the opportunity, resources and practical insights that helped shape the direction and quality of this work. Beyond the professional support, I would like to thank the team for making this experience truly enjoyable. Their kindness and humor created an atmosphere where learning and collaboration were not only effective but also fun. From insightful discussions to countless jokes shared along the way, every day at KPN was filled with energy and positivity, making this journey memorable on both a professional and personal level.

Finally, I want to thank my family, my girlfriend Samia and my friends for their unwavering encouragement, patience and support. Your belief in me has been a constant source of motivation.

To everyone who contributed in any way to this journey I want to take this opportunity to thank you.

Contents

1

Introduction

The relentless growth in mobile data traffic, driven by connectivity demands, upcoming applications such as the Internet of Things, augmented/virtual reality and the evolution towards 5G-Advanced and 6G networks, places unprecedented demand on network reliability and performance [1]. The second phase of 5G, 5G-Advanced aims to enhance current 5G capabilities and serves as a critical stepping stone towards the realization of 6G around 2028 [2]. Furthermore, the global 5G subscriptions has surpassed 1.6 billion, which accounts for 18% of total subscriptions by the end of 2023 and mobile data traffic which is approximately 143 EB per month, is projected to triple by 2029 [3]. These trends are amplifying the operational complexity and performance demands placed on telecom networks, pushing network operators to invest in automation, 5G-Advanced technologies and network efficiency strategies to remain competitive [4]. Modern mobile networks are becoming increasingly complex, comprising heterogeneous, multi-domain architectures including the Radio Access Network (RAN), Core Network, Transport Network and distributed management/control functions across centralized and edge deployments [5]. As mobile networks evolve to support complex architectures and next-generation applications, the associated increase in operational complexity elevates the risk of faults. A fault is the inability of an item to perform a required function, excluding that inability due to preventive maintenance, lack of external resources or planned actions [6]. This inability can stem from hardware failures, software issues or configuration mistakes. Due to tight interdependencies between domains, faults often propagate rapidly across domains, causing service degradation, security vulnerabilities or in worst case scenario full-scale outages. Each consequence could present a significant financial and reputational risks to Mobile Network Operators [7].

Traditionally, network fault management relies on alarms triggered by threshold violations in Key Performance Indicators (KPIs) within individual domains (e.g., RAN call drop rate, core network latency, transport packet loss), resulting in a largely reactive system [8]. This approach encounters several critical limitations, where action is often taken only after users have been affected by service disruptions. Correlated faults generate an overwhelming number of alarms due to error propagation or simultaneous detections obscuring the root cause. Additionally, troubleshooting within isolated domains ignores cross-domain dependencies, making root cause analysis slow and labor intensive. Lastly, fault detection and localization generally occur only after symptoms are observed, as traditional systems lack proactive or predictive capabilities and often rely on threshold-based alarm triggers [9].

While traditional fault management systems operate reactively, machine learning methods provide promise for proactive, scalable solutions. However, bridging the gap between detection and localization, especially across network domains, remains an open challenge. Predictive Fault Localization (PFL) refers to the task of forecasting the occurrence of faults and precisely identifying their originating locations (e.g., domains or elements) within the network topology before their effects propagate to users [10]. PFL extends classical fault localization by incorporating modeling, anomaly correlation across time and space (network domains or locations) and causality inference to anticipate and localize failures proactively. Achieving this goal necessitates the correlation of KPIs across multiple network domains, specifically RAN KPIs (e.g., signal strength, handover success, drop rate), Core Network KPIs (e.g., latency, throughput, processing load) and End-to-End (E2E) Service KPIs (e.g., user session success rate, latency) must be jointly analyzed to infer latent anomalies and anticipate fault propagation paths. Multi-domain KPI correlation is critical because faults often originate in one domain and propagate cascading effects to other domains, obscuring the root cause. For instance, a sudden traffic surge due to unbalanced distribution of traffic in the RAN could trigger processing overload in the Core Network's User Plane Function, manifesting as increased packet processing latency. If unresolved, this Core overload subsequently causes failures in E2E service KPIs, such as degraded session setup success rate or

increased application timeouts for end-users.

Machine learning–based methods, notably clustering and graph-based approaches, have been widely applied to detect anomalies in complex, high-dimensional data such as time-series and spatial patterns [11]. Furthermore, there have also been numerous studies that have explored ML for network fault detection and diagnosis, but often focusing on specific domains like RAN or core [12]. Particularly, 3GPP SA5 has introduced AI/ML models into the network management architecture through specifications like TS 28.104 and TS 28.105, formalizing training and inference functions for management systems [13]. Similarly, ETSI Experiential Networked Intelligence (ENI) defines closed-loop AI-driven network management frameworks with context-aware decision-making [14]. However, a significant research gap exists in developing and validating robust Predictive Fault Localization frameworks that effectively integrate and correlate heterogeneous KPIs across the key functional domains of RAN and Core, linked explicitly to the E2E service layer, to achieve precise fault localization with minimal lead time for proactive intervention. This challenge is further worsened by the limited availability of high quality labeled data, which also often requires resource-, time- or labour-intensive computational or experimental efforts. For example, accurate localization across domains, demands substantial expert input and is rarely feasible at the scale required for supervised machine learning [15]. Challenges include correlating low-level RAN/Core anomalies with user-impacting E2E symptoms, data heterogeneity, scale, minimal latency processing requirements, handling concept drift and establishing causally meaningful links between cross-domain KPI anomalies and specific fault types [16].

Problem Statement

The limitations of traditional reactive fault management and the practical challenges of acquiring sufficient labeled fault data create a significant barrier to deploying effective PFL in complex, multi-domain mobile networks. Unsupervised Anomaly Detection is defined as training without labeled samples, describing it as identifying data points that differ from the majority of observations by learning latent low-dimensional manifolds representing normal behaviour [17]. This is ideal for fault detection where labeled anomaly data is scarce. While unsupervised anomaly detection can identify deviations in KPI streams without labels, it can not localize root causes [18]. On the other hand, Reinforcement Learning (RL) has the potential to learn optimal fault mitigation policies through interaction. However, a key challenge is that RL traditionally relies on well-defined reward signals, which are often derived from labeled fault states or simulations, creating a dependence on scarce labeled data [19]. This leads to the research question to be addressed in this thesis: **How can unsupervised anomaly detection and reinforcement learning (RL) enable near real-time fault localization with minimal labeled data in mobile networks?**. While 2G is gradually being phased out, the focus is shifting toward 4G and 5G networks, which represent the future of connectivity. To address this, unsupervised learning techniques are used to detect anomalies in heterogeneous KPI time series across the RAN, Core Network and E2E service layers with the focus on 4G data. These anomalies act as early indicators of potential faults. Subsequently, offline reinforcement learning is used to infer fault locations by associating patterns of anomalies. An offline version is used due to the lack of a simulator or a digital twin.
The goal is to design a Predictive Fault Localization framework that:

- Leverages Unsupervised Learning to detect anomalous patterns in multi-domain (RAN, Core, E2E) KPI time-series data without/minimally relying on labeled fault examples.
- Utilizing Reinforcement Learning to utilize these detected anomalies as inputs for an RL agent, enabling it to learn policies for localizing the probable source of the fault.
- Minimizing Label Dependence to significantly reduce the reliance on scarce, costly and often imperfectly labeled historical fault data for training and operation.

Thesis Outline

This thesis is structured as follows: Chapter 2 offers insights into the principles and functioning of mobile networks. Chapter 3 provides a literature review that summarizes the current knowledge, advancements and challenges of the current anomaly detection and reinforcement learning. Chapter 4 details the implementation of the hybrid Predictive Fault Localization framework. It explains how unsupervised anomaly detection and reinforcement learning are integrated and applied to multi-domain KPI data.

Chapter 5 will show the test results. Chapter 6 summarizes the key findings, reflects on the work in this thesis and outlines potential directions for future research in predictive fault management for mobile networks.

2

Mobile Networks

Long-Term Evolution (LTE), also known as 4G, represents a fundamental architectural shift in mobile telecommunications, transitioning from the circuit-switched paradigms of 2G/3G to an all-IP, packet-only network. Standardized primarily by the 3rd Generation Partnership Project (3GPP), LTE was designed to achieve higher network performance, increased efficiency and overall network capacity [20]. This is achieved by reducing the latency since the Evolved Node B (eNB) is directly connected via S1 interface to the Evolved Packet Core (EPC) and also faster handover thanks to direct connectivity between eNBs via X2 interface [21]. This chapter provides a technical overview of the LTE system architecture, including its functional split between the control and user planes and the fundamental protocols that govern data and signaling flow.

The LTE system is divided into two primary subsystems: the Evolved Universal Terrestrial Radio Access Network (E-UTRAN) and the Evolved Packet Core (EPC). The Access Network connects end-user devices (like mobile phones, computers, IoT devices) to the core network. The EPC is the central brain and backbone of the LTE network, responsible for authentication, mobility management, session establishment and providing connectivity to external packet data networks e.g., the internet. The LTE Network Architecture can be seen in Figure 2.1 [22] and contains several components:

- **User Equipment (UE)**: The UE is the end-user device, such as a smartphone, tablet, or IoT module. It contains the Subscriber Identity Module (SIM), now embedded as an eSIM, which holds the subscriber credentials used for authentication with the core network.
- **Evolved UTRAN (E-UTRAN)**: The E-UTRAN consists solely of eNBs, which are the base stations. It provides wireless communication between mobile devices and the core network. A key innovation in LTE was the elimination of the centralized radio network controller (RNC) found in 3G Universal Mobile Telecommunications System (UMTS) networks. In LTE, the eNBs are intelligent and interconnected, assuming all radio-related functions, including radio resource management, handover decisions and header compression. This distributed "flat architecture" minimizes latency for user data transmission [23].
- **Mobility Management Entity (MME)**: The central control node for the access network. It handles signaling related to UE tracking, paging, authentication and bearer activation/deactivation.
- **Serving Gateway (SGW)**: Acts as a mobility anchor for the user plane during inter-eNodeB handovers, which is the process of transferring an active connection from one base station to another, so the user maintains uninterrupted service while moving. It routes and forwards user data packets between the eNodeB and the PGW.
- **Packet Data Network Gateway (PGW)**: The point of exit and entry for user traffic towards the internet. The PGW performs key functions such as IP address allocation for the UE, policy enforcement, and charging support.
- **Home Subscriber Server (HSS)**: A central database that contains user-related and subscription-related information. It performs authentication and authorization of the user and provides the user's profile to the MME.
- **Packet Data Network (PDN)**: A network that a mobile device uses to access data services like the internet.

The interconnection of these components form the foundation for all LTE communication, which is separated into two distinct planes: the Control Plane and the User Plane. A fundamental design

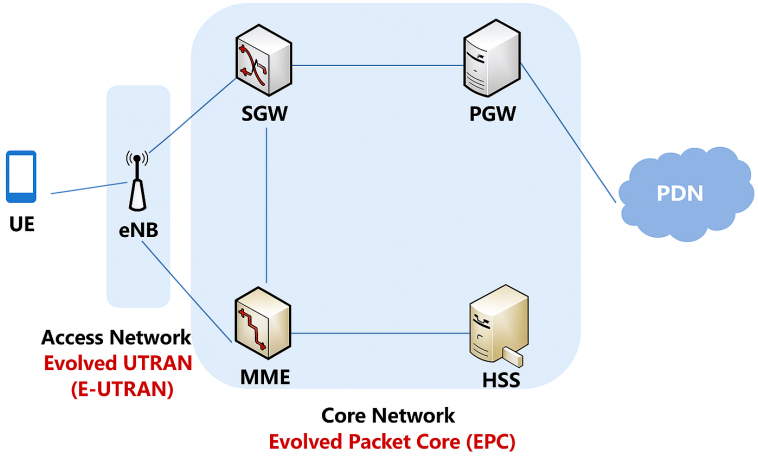

Figure 2.1: Overview LTE Network

principle in LTE is the clear separation between the Control Plane (C-Plane) and the User Plane (U-Plane). This separation enhances scalability, allows for independent optimization of signaling and data traffic and supports the transition towards network virtualization and the flexible architectures needed for 5G [24].

2.1. Control Plane

The Control Plane is responsible for the exchange of signaling messages that manage the network itself. It establishes the UE's initial connection and verifies its identity. Furthermore, it sets up, manages and terminates the logical data paths (bearers) required for user traffic with specific Quality of Service (QoS) guarantees and manages the location of the UE and executes the handovers between cells, a geographic area served by a single base station. The typical flow of control plane signaling for a session setup is illustrated in Figure 2.2.

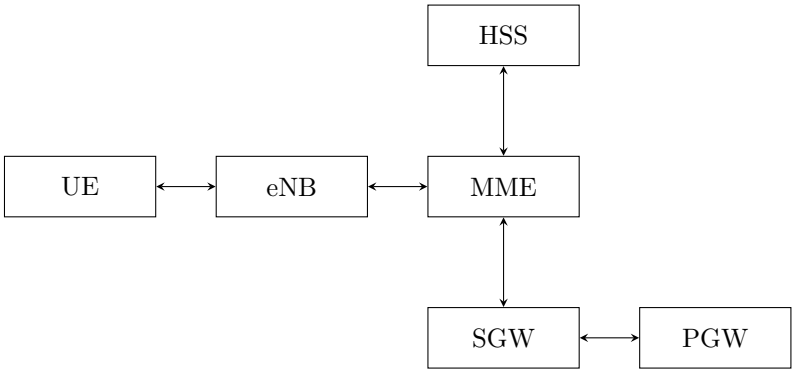

Figure 2.2: Control Plane

2.2. User plane

The User Plane, also known as the Data Plane, carries the actual user-generated data traffic (e.g., web pages, video streams, voice packets). Its design prioritizes high throughput and low latency. The data packets are tunneled between the network entities using the GPRS Tunneling Protocol for the user plane (GTP-U). This tunneling protocol ensures that the user's IP packets are seamlessly transported through the EPC, with the SGW and PGW acting as the tunnel endpoints. The path of user data

through the network is depicted in Figure 2.3.

Figure 2.3: User Plane

2.3. Key Performance Indicators

The health and performance of an LTE network are continuously monitored through counters. Key Performance Indicators (KPIs) can be calculated from the counters, e.g. the incoming handover success rate of a cell can be calculated using two counters, which are the all successful incoming handovers divided by all the attempted incoming handovers. Counters are therefore raw measurements collected, while KPIs are calculated metric used to evaluate how well a network, service or process is performing. By monitoring KPIs, network operators are able to analyze and optimize network operations through identifying performance bottlenecks, plan capacity upgrades and improve overall service quality, ensuring a better experience for subscribers. Therefore, KPIs help to understand how well the network meets customer expectations and technical standards. In mobile networks, KPIs can cover various aspects such as network availability, coverage, capacity and user experience. For example:

- **Accessibility KPIs**: Measure the network's ability to grant users access. Failures in for example RRC Connection Setup Success Rate could lead to issues in the control plane of MME or radio interface.
- **Retainability KPIs**: Measure the network's ability to maintain an active connection e.g. Session Drop Rate.
- **Performance KPIs**: Measure the quality of the service once established. e.g. Throughput (uplink and downlink).
- **Mobility KPIs**: Measure how effectively the network handles user movement, e.g. Handover Success Rate between cells.

2.4. 5G

The Fifth Generation of mobile networks (5G) represents the evolution beyond LTE, designed to meet the growing demands for ultra-fast data rates, extremely low latency and increasing connectivity. The objective of 5G is to support three main services [25]:

- **Enhanced Mobile Broadband (eMBB)**: Provides higher data rates and improved capacity to handle large volumes of traffic in dense urban environments.
- **Ultra-Reliable Low-Latency Communications (URLLC)**: Enables real-time applications that demand extreme reliability and very low latency, such as autonomous vehicles.
- **Massive Machine-Type Communications (mMTC)**: Supports the connection of large quantities of low-power IoT devices with efficient signaling and minimal energy consumption.

2.4.1. Architecture Overview

The 5G architecture builds on the concepts of LTE but add a more flexible and modular design including two main components:

- **5G Core (5GC)**: A fully virtualized and service-based architecture (SBA), where each network function (NF) communicates with others via standardized APIs using HTTP/2, which is the successor of HTTP/1.1 to make websites faster and more efficient, over a service bus. This approach enhances scalability, interoperability and facilitates cloud-native deployment with functions:

  - **Access and Mobility Management Function (AMF)**: Handles registration, connection management, and mobility management.
  - **Session Management Function (SMF)**: Responsible for session establishment, QoS enforcement, and IP address allocation.
  - **User Plane Function (UPF)**: Handles user data forwarding, similar to the SGW/PGW in LTE, but is designed for distributed deployment close to the user edge to minimize latency.
  - **Unified Data Management (UDM)**: Equivalent to the HSS in LTE, managing subscription and authentication data.
  - **Network Slice Selection Function (NSSF)**: Assigns network slices to users or services based on their requirements.
- **5G New Radio (NR)**: The new air interface that operates across a wide range of frequency bands, from sub-1 GHz to millimeter wave (mmWave) spectrum (>24 GHz). NR introduces technologies such as beamforming and massive MIMO (Multiple input multiple output), which exploits the spatial domain to improve the coverage, capacity and user throughput of mobile networks [26] and therefore allowing dynamic adaptation of transmission parameters to meet varying service demands.

2.4.2. Network Slicing and Virtualization

One of the defining innovations in 5G is **Network Slicing**, which allows the physical network infrastructure to be partitioned into multiple virtual networks (or slices), each tailored to a specific service or customer requirement. For instance, one slice may be optimized for low-latency industrial control, while another is designed for high-throughput video streaming. Each slice operates independently in terms of control, management and resource allocation [27].

This capability is enabled through the integration of **Network Function Virtualization (NFV)** and **Software-Defined Networking (SDN)**, which decouple network functions from proprietary hardware, allowing them to run as software instances in cloud environments. This flexibility accelerates network deployment, supports dynamic scaling and enables rapid introduction of new services.

2.4.3. 5G and LTE Interworking

During the early phases of 5G deployment, Non-Standalone (NSA) architecture is commonly used, where the LTE Evolved Packet Core (EPC) continues to manage control-plane functions while 5G NR provides additional radio capacity. In contrast, the Standalone (SA) architecture introduces a full 5G Core (5GC), enabling complete independence from LTE and supporting new 5G-native features such as network slicing and service-based connectivity.

2.4.4. Implications for Network Performance Monitoring

As 5G networks evolve, Key Performance Indicators (KPIs) must adapt to reflect new dimensions of network performance. In addition to traditional measures such as throughput, latency and handover success, 5G introduces KPIs related to:

- **Network Slice Utilization and SLA Compliance**
- **Edge Computing Latency**
- **Beamforming Efficiency and Signal Quality**
- **End-to-End Service Experience Across Multi-Domain Networks**

In summary, 5G represents a paradigm shift from the static, hardware-centric architecture of LTE to a dynamic, software-driven ecosystem. Its flexibility, combined with cloud-native design and intelligent automation, makes it the foundational technology for future digital transformation across industries.

3

Literature Review

This chapter reviews the state of the art in Unsupervised Anomaly Detection and Reinforcement Learning and discusses how these two domains increasingly overlap. While Anomaly Detection focuses on identifying data patterns that deviate from expected behaviour, RL offers a framework for making adaptive decisions in response to those irregularities.

3.1. Unsupervised Anomaly Detection

Unsupervised anomaly detection has evolved from traditional statistical models to modern deep learning architectures capable of modeling complex, high-dimensional data. Unsupervised learning refers to a model that is trained on data, that has no labels or predefined outcomes like supervised learning. Despite these advancements, the key challenge remains unchanged: identify data points, sequences or patterns that deviate from normal behaviour with no supervision.

3.1.1. Classical and Machine Learning Methods

Classical statistical anomaly detection methods often assume data follow a certain distribution, e.g., Gaussian, and treat samples with low probability density as anomalies [11]. These methods can be classified into three types:

- **Distance-based methods**: Anomalies are points far from their nearest neighbours. A data point is considered anomalous if its average distance to its k nearest neighbours is significantly larger than that of most other points, e.g., k-Nearest Neighbours (k-NN) [28].
- **Density-based methods**: These compare the local density of a point to the densities of its neighbours. Points with significantly lower density are flagged as anomalies, e.g., Local Outlier Factor (LOF) [29].
- **Isolation-based methods**: Explicitly isolate anomalies by randomly selecting features and split values. Anomalies require fewer splits to be isolated, e.g., Isolation Forest [30].

These methods remain effective for low-dimensional or tabular data, but often struggle with complex temporal or spatial structures.

3.1.2. Deep Learning Architectures

Deep learning has revolutionized anomaly detection by enabling automatic feature learning from complex and high-dimensional data. The following architectures dominate modern unsupervised anomaly detection research:

- **Autoencoders (AEs)**: Learn a compressed representation (latent space) of the data. The underlying assumption is that the model, trained predominantly on normal data, will reconstruct well but will fail to reconstruct anomalies. This will lead to a high reconstruction error [31].
- **Generative Adversarial Networks (GANs)**: Train a generator to reproduce “normal” data and a discriminator to distinguish real from generated data. Anomalies can be detected by reconstruction error or discriminator output [32].
- **Self-Supervised Learning (SSL)**: Use anomaly detection as a supervised learning task by creating ”pretext tasks” from the normal data itself, e.g. predicting geometric transformations, masking and predicting patches. A model’s inability to solve these tasks effectively can indicate an anomaly [33].
- **Graph Neural Networks (GNNs)**: Handle relational data such as social networks, financial transactions or IT infrastructures by learning node embeddings from graph structures. Nodes

with embeddings that deviate from their neighbours are marked anomalous [34].

3.1.3. Anomaly Detection in Telecom Networks

In mobile networks, deep and graph-based unsupervised anomaly methods have been applied for detecting traffic surges, configuration errors and abnormal performance patterns [35]. However, these approaches often operate within a single domain and is therefore limited in cross-domain correlation. Furthermore, even when anomalies could be detected, the origin of the fault could not always be inferred [36].

3.1.4. Current Limitations

Despite developments, several limitations of unsupervised anomaly detection in mobile networks are:

- Lack of labeled ground truth for validation and interpretability.
- Static models fail under non-stationary network conditions or concept drift. This is the phenomenon where the statistical relationships between input data and the target variable change over time. This could cause a machine learning model's predictions to become less accurate.
- The majority of the unsupervised anomaly detection are passive, identifying anomalies, but not linking them to root causes or adaptive actions.

These limitations motivate combining unsupervised anomaly detection with reinforcement learning for adaptive reasoning and localization.

3.2. Reinforcement Learning

RL studies how an agent learns to interact with an environment by taking actions that maximize cumulative reward. Formally, RL problems are typically modeled as Markov Decision Processes (MDPs) [37]. Modern RL methods can be categorized by their learning strategy: Reinforcement learning (RL) focus on how an agent should take actions in an environment to maximize a cumulative reward. RL methods can be classified into three types:

- **Value-Based Methods**: Learn an optimal action-value function $Q(s, a)$, which estimates the expected cumulative reward for taking action a in state s. The agent selects the action with the highest Q-value, e.g. Deep Q-Learning [38].
- **Policy-Based Methods**: Directly learn a policy $\pi(a|s)$ without estimating value functions. Policy-Based methods are more stable and effective for continuous action spaces, e.g. Proximal Policy Optimization (PPO) [39].
- **Actor-Critic Methods**: Hybrid approaches that combine value-based (the critic: evaluates the action) and policy-based (the actor: decides the action) methods. This is the foundation for most state-of-the-art algorithms, e.g. Isolation Forest [40].

3.2.1. Batch RL

Offline RL, also known as batch RL, learns policies solely from pre-collected data, without environment interaction. This is essential for complex, large-scale real-world infrastructures, such as mobile networks or aerospace systems, where applying directly to the systems is unsafe or infeasible and a digital twin is not available. Recent methods improve stability by regularizing learned policies to remain within the support of the dataset [41].

3.2.2. Challenges in RL

While RL operate well at adaptive control, several issues limit its application to anomaly detection:

- **Sample inefficiency**: Deep RL often requires millions of interactions to converge.
- **Safety and stability**: Exploration can lead to catastrophic states, which arehighly undesirable or unsafe situations that may cause system failure, e.g., a robot falling off a ledge while learning to navigate.
- **Reward design**: Defining meaningful reward signals for detection or safety tasks is difficult.

4 Methodology

This chapter describes the methodology used to develop and evaluate a PFL framework that combines data processing, unsupervised anomaly detection and reinforcement learning for fault localization in multi-domain mobile networks. The goal is to localize faults across the RAN, Core Network and E2E service domains with minimal reliance on labeled fault data. The full pipeline can be seen in Figure 4.1.

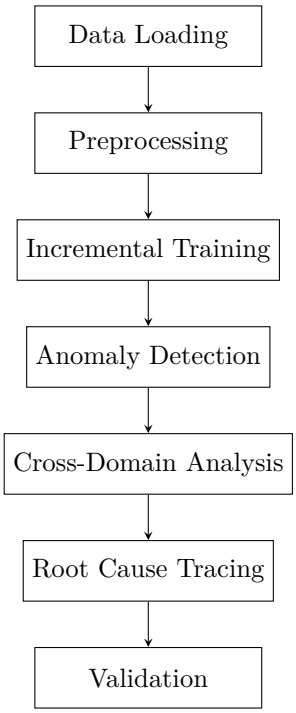

Figure 4.1: High-level Pipeline of the Root Cause Analysis Process

To address the complexity of real-world telecom environments and considering the limitations of access to data due to the internship setting at KPN, where direct access to real-time data and KPI databases is not permitted for security and compliance reasons. KPI datasets are retrieved, saved and securely transferred by the supervisor at KPN. All KPI data is recorded at either 5- or 15-minute intervals. The smallest consistent timestamp available across the dataset represents the minimum interval, which is 15 minute interval. The data is downloaded and provided in .csv and .parquet format. Time in the data is always measured in milliseconds (ms). Although, this setup restricts real-time experimentation and system integration, it provides a realistic prototype.

4.1. Anomaly Detection

The anomaly detection consists of training, testing and tuning the machine learning components of the PFL framework using historical data. This phase is performed using daily historical KPI datasets

provided by KPN and consists of three primary objectives. One is to preprocess the data before entering the unsupervised anomaly detection model. The second is to train an unsupervised anomaly detection model that can recognize abnormal behaviour in multi-domain KPI time-series data. The last objective is to use detected anomaly patterns as input for training a reinforcement learning agent capable of localizing faults within the network topology. The datasets are from the KPN network in the Netherlands, excluding any roaming traffic.

4.1.1. Data preprocessing

The preparation of KPI data for unsupervised anomaly detection is a critical process in order to have clean and correct data for the later processes. This includes four key steps: combining data, filtering, normalization and window creation. Each step addresses the data quality and modeling requirements to ensure a robust anomaly detection.

The first step filters the previous day's file to include only data from the last 45 minutes, which is then combined with the current day's file to create a 24.75 hour dataset for window generation. The second step is to filter counters across data files that are needed to calculate KPIs. Initially, the focus is on selecting 13 KPIs. This limited selection allows easier verification of functionality in the later stages. More KPIs will be incorporated when the system functions correctly to scale up. The chosen KPIs are listed in Table 4.1 and were selected based on discussions with domain experts at KPN and include commonly used indicators of network faults, such as RRC reestablishment success rate. E2E have been split into 2 parts E2E PGW SGW and E2E cell to reduce the size of the data. The E2E Cell dataset represents measurements at the cell level, capturing performance indicators specific to individual radio cells. The E2E PGW/SGW dataset represents the userplane/dataplane routing path, indicating through which Packet Gateway and Serving Gateway the traffic was routed. Furthermore, to ensure temporal consistency across all data, timestamps originally provided in epoch format are transformed to UTC and then to Europe/Amsterdam timezone, while timestamps already expressed in UTC are directly converted to Europe/Amsterdam without intermediate processing. This conversion aligns all measurements to local network operation time and prevents misalignment caused by daylight saving time changes or mixed timezone representations during window generation and aggregation. Some data arrives at varying intervals, some KPIs are recorded at 5-minute intervals rather than the required 15-minute intervals. In these cases, the data is aggregated accordingly. For example, timestamps like 00:00, 00:05 and 00:10 are grouped under the 15-minute 00:00 timestamp, 00:15, 00:20 and 00:25 are grouped under the 15-minute 00:15 timestamp. The third step includes precomputing global statistics for normalizing to prevent the boundary effect. The next step is to normalize the KPIs, which is crucial for two primary reasons. First, both unsupervised anomaly detection model and reinforcement learning agent rely on gradient-based optimization and distance metrics. Without normalization, high magnitude features could possibly dominate low magnitude ones and therefore leading to a biased model [42]. Second, normalization improves training efficiency by accelerating convergence and reducing the risk of gradient explosion or vanishing gradient [43].

The chosen normalization method is Robust Scaler. Robust Scaler normalization is a normalization technique that uses the median and the interquartile range (IQR) to normalize features. The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set using the formula shown in Equation 4.1[44].

$$x_{scaled} = \frac{x - median}{IQR} \tag{4.1}$$

The choice of Robust Scaling is motivated by several considerations:

1. **Resistance to outliers**: Mobile network data is known as noisy and could consist of legitimate outliers, such as handover failure due to an event which is defined as a temporary surge in network activity. This could occur during large gatherings, like a festival, sports match or concert, where the sudden increase in users causes unusual traffic patterns. Such anomalies reflect real-world events rather than system faults. The Min-Max scaling will be skewed by these points. Z-Score

Category	Metric	Description
Core	gxccrccaaveragelatency	The average latency between receiving a Gx CCR (including CCR-I, CCR-U, and CCR-T) and sending a corresponding Gx CCA (in ms) *
	rxaargxraraveragelatency	The average latency between receiving an Rx AAR (including AAR-I and AAR-U) and sending a corresponding Gx RAR (in ms) **
RAN	rrc_connection_success_rate	RRC connection success rate
	rrc_reestablish_success_rate	RRC reestablishment success rate
	cce_utilization	Control Channel Element utilization
	prb_dl_utilization	Downlink Physical Resource Block utilization
	prb_ul_utilization	Uplink Physical Resource Block utilization
E2E	core_to_ext_rtt	Round-trip time from core to external network (in ms)
	core_to_ue_rtt	Round-trip time from core to UE (in ms)
	out_of_order_packets_downlink_rate	Downlink out-of-order packet rate
	out_of_order_packets_uplink_rate	Uplink out-of-order packet rate
	average_downlink_throughput	Average user downlink throughput (in Mbps), representing the average data rate successfully received by UEs over time
	average_uplink_throughput	Average user uplink throughput (in Mbps), representing the average data rate successfully transmitted by UEs over time

Table 4.1: Chosen initial KPIs

* Gx represents the interface between the Policy and Charging Rules Function (PCRF), which makes real-time policy and charging decisions for subscriber data sessions, and the Policy Enforcement Point (usually PGW), which handles the policy control of data sessions. CCR stands for Credit-Control Request, which is a request sent by the PGW to PCRF over Gx. CCA stands for Credit-Control Answer, which is the response from PCRF back to PGW.

** Rx represents the interface between the PCRF and the Application Function (AF), which is often used for Quality of Service (QoS) for e.g., Voice over Internet Protocol (VoIP). AAR stands for Authorization-Authentication Request, which is sent by the AF to PCRF over Rx. RAR stands for Re-Authorization Request, which is sent by the PCRF to AF to update session parameters.

will also be influenced. However, Robust Scaler, which uses median and IQR is more resistant to outliers.

2. **Non-skewed**: Network KPIs are often right-skewed (non-gaussian) [45]. There are KPIs with no negative values, but could in turn have very high values. The Z-Score assumes a relatively normal distribution for the standard deviation (SD), however the Robust Scaler does not make this assumption.
3. **Preserves information about anomalies**: The goal is to find anomalies. If an outlier is so extreme that it pulls the min/max or mean/SD, this would make other smaller anomalies harder to detect. By using robust statistics, the Robust Scaler ensures that true anomalies remain "far away" from the normalized center, making them easier for your model to identify.

The final step in preparing the KPI data is the creation of time windows, which is crucial for capturing temporal patterns and context. Single 15-minute KPI snapshots lack the necessary context to identify trends or evolving issues. Many types of faults do not appear as isolated anomalies, but emerge gradually over time. For example, an increase in latency that eventually leads to a drop in throughput. Viewing

KPIs over time helps revealing how issues develop and escalate, which is critical for pinpointing root causes and improving fault localization. Therefore, a sliding window approach is used. Each input to the model includes multiple consecutive 15-minute intervals, covering one hour of historical data. This allows the system to observe how metrics change over time and identify sequences that deviate from normal behaviour. It also transforms the data into a more meaningful representation for anomaly detection and decision-making tasks.

4.1.1.1. Challenges

Several practical challenges were addressed during the data preparation. This includes:

- **Big Data Volume**: With datasets exceeding 100 million rows, conventional tools are ineffective, such as pandas [46]. Therefore polars [47] is used, which is a fast, memory-efficient and parallelized DataFrame library built on Rust and Apache Arrow, designed to handle massive data. Furthermore, .parquet is used for the loading and saving data instead of .csv due to a compressed, columnar storage format that reduces disk space and speeds up read/write operations. Since directly viewing such large datasets and parquet is impractical, a small script is written to sample and display a subset of the data for verification purposes.
- **Timestamp generation for sparse data**: Certain counters, such as handovers, only record a timestamp when an event occurs. To ensure a continuous, consistent 15-minute interval for the unsupervised anomaly detection, a complete timestamp is generated setting to 0 or 100 accordingly. When no event occurs within an interval, the corresponding KPI is filled with default values (e.g., a handover success rate of 100). These generated timestamps are then left-joined with the original data.
- **25 hours data**: Every dataset consists of 25 hours instead of 24 hours. This choice is made due to the last hour of window creation cannot be completed. For example, a window starting at 23:15 requires data points at 23:15, 23:30, 23:45, and 00:00 of the next day. Since the next day's data is unavailable, an extra hour is included to ensure all windows for the current day can be constructed for the window creation. Prior to window generation, the first hour of the following day is excluded from further analysis. However, this was later disregarded due to the bullet point **Forward or Backward looking**. Instead 45 minutes of the prior day is filtered and combined to the current date dataset.
- **Data volume**: Retrieving the raw data from the platform is extremely time consuming due to its size. For example, one day of handover dataset alone amounts to approximately 20 GB. To address this, the handover attempts are aggregated per base station per cell per day. Subsequently, the base stations are ranked according to their contribution to the total number of handover attempts and only the base stations that cumulatively account for up to 90% of the total attempts are retained. This approach ensures that the dataset remains computationally manageable while preserving sufficient data quality for analysis. On top of that the handover dataset has been delivered as .parquet instead of .csv to reduce the size even more.
- **Boundary effect**: A 'Boundary Effect' refers to the bias that can occur in spatial analysis when areas close to boundaries, such as national or ocean boundaries, significantly differ from the rest of the region [48]. In this case, the boundary would be the end and beginning of the day. This effect occurs because the last three time windows of one day may have different values than the initial three windows of the following day. This discrepancy arises from the use of normalization per day, causing the last few windows of one day not to align with the beginning of the next as each uses its own day for normalization. To mitigate this issue, data is filtered and the necessary components such as median and IQR are calculated for multiple days, while excluding the first hour of the next day of the 25 hour dataset. This ensures that the discrepancy between days are removed.
- **Forward or Backward looking**: During the window generation for time series anomaly detection, both forward- and backward-looking approaches were evaluated. In the first approach, windows were created using future-shifted values (e.g., t, t+1, t+2, t+3), corresponding to a forward-looking sequence. However, this approach is unsuitable for real-time detection because future information is unavailable when processing live data streams. Moreover, using future values might cause data leakage, which is when a model unintentionally learn patterns that depend on

information that would not yet exist in a production or monitoring environment. This will lead to an extreme optimistic performance during training and evaluation, but fails under real operating conditions. Furthermore, such forward-looking models violate the temporal causality inherent in time series data, as anomalies must be detected based solely on information available up to the current timestamp. Therefore, the method was adapted to a backward-looking structure, where only historical values $(t-3, t-2, t-1, t)$ are used to represent the temporal context. This ensures that anomaly detection remains causal and applicable in real-time environments.

4.1.2. Unsupervised Anomaly Detection

Selecting the most suitable unsupervised anomaly detection is a crucial process, as it constitutes one of the core components. Based on the requirements, several anomaly detection models can be excluded. The data consists of KPIs recorded at 15-minute intervals. On top of that, sliding windows are used to capture temporal patterns and context. Consequently, one of the requirements states that the model must be capable of learning sequences and trends over time. Another requirement is that since there is no labeled fault data available, supervised models that require pre-classified data, such as what "normal" and "faulty" states are, are ruled out. Therefore the chosen model must operate in an unsupervised manner. Moreover, all features are normalized using robust scaling, which is beneficial for Gradient-based models including, neural networks and LSTMs as it leads to faster convergence and more stable training.
Taking these requirements into account leads to one model that satisfies the best based on the requirements, which is the **LSTM Autoencoder**. Long Short-Term Memory (LSTM) networks are a specialized type of Recurrent Neural Network (RNN) that extend conventional RNNs by addressing their limitations in modeling long-term dependencies, enabling them to learn long-term patterns in sequence data. This capability makes LSTMs particularly suitable for a variety of sequence modeling tasks, including time series analysis, natural language processing and other applications that require capturing complex temporal dynamics [49]. Autoencoders are unsupervised neural architectures that are capable of learning to reconstruct input data in an unsupervised manner. The input is first compressed (encoded) into a lower-dimensional latent space and then recreated (decoded) with minimal error. A latent space is a lower-dimensional representation of data where the essential features and patterns are captured, while less important details are discarded. In this structured space, similar inputs are placed close together. For example, in an autoencoder trained on faces, faces with similar attributes (like smiling vs. neutral, or glasses vs. no glasses) may cluster together in latent space. Through this encoding and decoding process, autoencoders effectively learn to capture the most essential features of the data while discarding noise and redundancy. Therefore, autoencoders are widely used in unsupervised learning scenarios for anomaly detection, removing noise, feature learning or data compression. The model learns a compressed representation of "normal" KPI behaviour during training. During testing, when a new time-series window is presented, the LSTM Autoencoder attempts to reconstruct it. If the pattern is consistent with the learned normal behaviour, the reconstruction error will be low. Conversely, if the input window contains unseen anomalous patterns, the reconstruction error will be significantly higher, allowing anomalies to be flagged.

4.1.2.1. Alternative Models

Other anomaly detection models were evaluated but considered less suitable:

- **Isolation forest / One-class Support Vector Machine (SVM)**: Effective unsupervised models for static, non-sequential data, but unable to capture temporal dependencies across windows.
- **Vector Autoregression (VAR)**: Time-series models, but generally utilized for forecasting and may be less flexible than a neural network approach for capturing complex, non-linear patterns across multiple KPIs.
- **Clustering-based Approaches**: Techniques such as k-means can group similar data points, but do not naturally account for sequential context, therefore making them unsuitable for temporal anomaly detection.

4.1.2.2. Implementation

The implementation of the LSTM Autoencoder consists of several key components that work together to enable comprehensive anomaly detection capabilities.

4.1.2.2.1 Data Preparation and Temporal Encoding

The input data of the anomaly detection are normalized windowed data described in paragraph 4.1.1. Each domain's time-series data is structured as sliding windows of length 4, representing consecutive 15-minute intervals. For a dataset with features $F = f_1, f_2, ..., f_n$, the input at time t is represented as:

$$X_t = [x_{t-3}, x_{t-2}, x_{t-1}, x_t] \tag{4.2}$$

where each $x_i \in \mathbb{R}^n$ represents the feature vector at time i. In the implementation lag indices are used and indicate the positions of observations within a time window. lag_0 refers to the current time step, whereas lag_1, lag_2, and lag_3 correspond to measurements taken 15, 30, and 45 minutes earlier, respectively.
To capture daily patterns in network traffic, temporal features are encoded cyclically using sine and cosine transformations shown in Formulas 4.3, 4.4, 4.5, 4.6 [50]. Cyclic variables are variables that have a repeating pattern over a defined interval, such as number of hours in a day or number of days in a week. These variables can be challenging in machine learning due to the cyclical nature that can not be captured well using simple numerical representation. This is an important issue many machine learning models interpret numerical distance linearly. Without cyclic encoding, the model would treat adjacent times (e.g., hour 23 and hour 0, or minute 59 and minute 0) as far apart numerically even though they are adjacent in reality. This artificial discontinuity can degrade the model's ability to learn smooth temporal patterns, leading to worse generalization and less accurate predictions. By transforming cyclic variables, the periodic structure is preserved, implying values that are close in time are also close in the transformed feature space. This in turn will allow models to capture daily and weekly rhythms in network traffic, such as peak usage during working hours or reduced activity on weekends.

$$hour_{sin} = sin(\frac{2\pi \cdot hour}{24}) \tag{4.3}$$

$$hour_{cos} = cos(\frac{2\pi \cdot hour}{24}) \tag{4.4}$$

$$minute_{sin} = sin(\frac{2\pi \cdot minute}{60}) \tag{4.5}$$

$$minute_{cos} = cos(\frac{2\pi \cdot minute}{60}) \tag{4.6}$$

Additionally, categorical temporal features are included to identify operational periods to allow the model to learn normal behaviour patterns, resulting in six temporal features:

- Peak hour indicator: $I_{\text{peak}} = 1$ if hour $\in [8, 18]$, else 0
- Night hour indicator: $I_{\text{night}} = 1$ if hour $\in [0, 6]$, else 0

4.1.2.2.2 Model Architecture Specification

The autoencoder architecture has different configurations for various data characteristics shown below:
Core domains

- Input dimension: $d_{\text{input}} = 7$ (1 original feature + 6 temporal features)
- Hidden dimensions: $[32, 16]$
- Dropout rate: 0.1

E2E and RAN domains

- Input dimension: $d_{\text{input}} = 16$ (10 original features + 6 temporal features for E2E) or $d_{\text{input}} = 11$ (5 original features + 6 temporal features for RAN)
- Hidden dimensions: $[64, 32, 16]$
- Dropout rate: 0.2

The encoder processes the input sequence through successive LSTM layers with the final hidden state serving as the latent representation. The decoder reconstructs the sequence from this representation, with the objective of minimizing the reconstruction error:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{T \cdot n} \sum_{t=1}^{T} \sum_{i=1}^{n} \left(x_{t,i} - \hat{x}_{t,i}\right)^2 \tag{4.7}$$

where T is the sequence length (4), n is the number of original features, $x_{t,i}$ is the actual value at time t, and $\hat{x}_{t,i}$ is the reconstructed value.

4.1.2.2.3 Training Configuration

To effectively train the LSTM Autoencoder, several hyperparameters must be set:

- **Epochs**: An epoch refers to one complete pass of the entire training dataset through the network. Multiple epochs are necessary for the model to gradually refine its parameters. However, too few epochs may result in underfitting, whereas excessive epochs risk overfitting to noise. For this reason, EarlyStopping and ReduceLROnPlateau have been included. EarlyStopping stops training when the model stops improving and therefore prevents overfitting and saves time by stopping training early when further epochs do not contribute anymore. ReduceLROnPlateau reduces the learning rate when the model stops improving, this helps the model converge better by slowing down learning when progress stalls. The chosen epoch is 75 for the first day and 50 for the remaining days.
- **Batch Size**: Training data are divided into smaller batches and the model parameters are updated after processing each batch. Smaller batch sizes introduce stochasticity. A small batch is a subset of the full dataset and therefore only an approximation of the true gradient computed over the full dataset. As a result, updates are noisy, which can help the model explore the parameter space more effectively, escape shallow local minima, avoid overfitting and improve generalization. Smaller batches increase this noise, making updates less stable but potentially more robust. While larger batches yield more stable but memory intensive training. This was chosen to be 2048 and 4096 for larger datasets.
- **Validation split**: A validation split is the portion of the dataset that is excluded from training and used to evaluate the model during training. It helps monitor generalization, tune hyperparameters and detect overfitting. The model never learns from this data, so performance on the validation set reflects how well it may perform on unseen data. This is set at 0.1.
- **Loss Function**: The reconstruction error is measured using Mean Squared Error (MSE), which quantifies the deviation between the input sequence and its reconstruction. Lower MSE values indicate accurate reconstruction of normal patterns, as shown in Equation 4.8.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y}_i)^2 \tag{4.8}$$

  where

  - MSE is the mean squared error,
  - n is the total number of data points,
  - y is the predicted value,
  - y_i is the actual (true) value of the i-th observation.
- **Learning Rate**: The learning rate controls the magnitude of weight updates during optimization. A balanced learning rate is crucial: large values may cause divergence, while small values may slow convergence or trap the model in local minima.
- **Early stopping**: Stops training when validation loss no longer improves. Early stopping does not reset when the learning rate changes. Therefore, early stopping may trigger even if the learning rate has not been reduced the maximum number of times. This is set at patience 3 with

best weights restoration. When early stopping is triggered, the model automatically reverts to the weights from the epoch with the lowest validation loss, ensuring optimal model performance without overfitting.

- **Hidden dimensions**: The hidden layer sizes determine the capacity of the autoencoder to learn representations from the input data. For the Core domain, the input dimension is small ($d_{\text{input}} = 7$), so a two-layer architecture with hidden dimensions [32, 16] is sufficient to capture relevant patterns without overfitting. For E2E and RAN domains, the input dimensions are larger ($d_{\text{input}} = 16$ for E2E, $d_{\text{input}} = 11$ for RAN), so a deeper architecture [64, 32, 16] is chosen to gradually compress the higher-dimensional input into a meaningful latent representation. The decreasing pattern of the hidden dimensions ensures that the network compresses information while retaining important features, which is critical for reconstruction and downstream tasks.
- **Dropout rate**: Dropout is a regularization technique used to reduce overfitting, which occurs when a model learns the training data too closely, including its noise and random fluctuations, leading to poor performance on new, unseen data. Dropout works by randomly "dropping" a fraction of neurons during each training step, forcing the network to learn redundant and more robust feature representations rather than relying on any single neuron. This will increase the training loss, but are disabled during validation. In the Core domain, a small dropout rate (0.1) is used as the model is smaller and less probable to overfit. For the deeper and wider E2E and RAN networks, a larger dropout rate (0.2) is applied to more strongly regularize the network and prevent it from memorizing the training data.
- **Reduce Learning rate**: The learning rate of ran, coreGx and coreRx start at 0.0010, halved when there is no improvement with patience 2, with limit of not to drop below 1e-6. The same is applied for E2E, but with a starting learning rate of 5e-4.
- **Optimizer**: An optimizer is an algorithm that adjusts a neural network weight during training to minize the loss function. The chosen optimizer is **Adam**, which is a deep learning optimizer that is popular for its efficiency and speed in training neural networks. Adam stands for Adaptive Moment Estimation and is a combination of two other methods: Momentum, which uses past gradients to smooth updates and RMSProp, which adapts the learning rate for each parameter. Adam keeps track of both mean and variance of gradients and updates weights using the moving averages, leading to stable and fast convergence.

During training, there are 2 losses, loss and val_loss. Loss is the training loss. This measures how well the model is performing on the training data. Calculated after each batch or epoch during training. A lower training loss means the model is fitting the training data better. This is computed after each epoch. Val_loss is the validation loss. This measures how well the model is performing on the validation data. It is a better indicator of generalization, how well the model will perform on unseen data. If val_loss is much higher than loss, it may indicate overfitting, memorizing the training data but not generalizing well. Therefore, the following can be stated:

- **val_loss > loss**: The model is optimized using the training data and therefore typically achieves a lower loss on the training set. Since the validation data is unseen during training, a higher validation loss is expected. A consistently large gap between training and validation loss may indicate overfitting, where the model fails to generalize to unseen data.
- **val_loss < loss**: A lower validation loss than training loss can occur when the validation data is less noisy or easier to model than the training data. This behaviour is commonly observed during early training or when regularization techniques such as dropout are applied.

Therefore, the ideal scenario would be val_loss follows the trend of loss, where both losses should be decreasing. This indicates that the model's prediction are becoming more aligned with the actual values.

The training process uses several techniques to ensure robust learning:

- **Incremental Training**: Models are trained across multiple days, allowing continuous adaptation to evolving network patterns. The learning schedule reduces from 75 epochs for initial training to 50 epochs for subsequent days.

- **Adaptive Learning Rate**: An Adam optimizer with ReduceLROnPlateau scheduling dynamically adjusts the learning rate based on validation loss, with an initial learning rate of 1×10^{-3} for core domains and 5×10^{-4} for E2E domains.
- **Early Stopping**: Training halts when validation loss fails to improve for 3 consecutive epochs, preventing overfitting and reducing computational overhead.
- **Gradient Clipping**: Gradients are clipped to a maximum norm of 1.0 to ensure stable training, particularly important for LSTMs which can suffer from exploding gradients.

4.1.2.2.4 Anomaly detection

The anomaly detection does not rely on single reconstruction error, the proposed approach combines three different components, each computed independently per feature, to improve robustness against both transient noise and slowly evolving faults.

First, for each feature j, the current reconstruction error quantifies the instantaneous deviation between the observed value $x_{t,j}$ and its reconstruction $\hat{x}_{t,j}$ produced by the autoencoder. This metric reflects how well the model represents that feature at the current time step and serves as the primary indicator of anomalous behaviour:

$$E_{\text{current},j}(t) = \left(x_{t,j} - \hat{x}_{t,j}\right)^2 . \tag{4.9}$$

While the instantaneous error detects abrupt changes, it may be sensitive to short-lived fluctuations. To mitigate this, a contextual deviation error is introduced. This metric compares the current observation to the average of the last three time steps, incorporating short-term temporal consistency into the anomaly score for each feature:

$$E_{\text{context},j}(t) = \left(x_{t,j} - \frac{1}{3}\sum_{i=t-2}^{t} x_{i,j}\right)^2 . \tag{4.10}$$

The temporal evolution of the reconstruction error is considered through the trend degradation error. This component focuses on trend deterioration by taking the positive part of the increase in the current error relative to the previous time step while ignoring recoveries:

$$E_{\text{trend},j}(t) = \max\left(0,\ E_{\text{current},j}(t) - E_{\text{current},j}(t-1)\right). \tag{4.11}$$

The final anomaly score is computed as a weighted combination of the three error components. The weights were determined to prioritize instantaneous reconstruction quality while still accounting for contextual consistency and temporal degradation:

$$E_{\text{combined},j}(t) = 0.6\, E_{\text{current},j}(t) + 0.3\, E_{\text{context},j}(t) + 0.1\, E_{\text{trend},j}(t). \tag{4.12}$$

A sample at time t is flagged as anomalous if any of its features exceeds a feature-specific, hour-adaptive threshold derived from the distribution of $E_{\text{combined},j}$.

4.1.2.2.5 Hour Based Adaptive Thresholding

To account for recurring daily patterns in network behaviour, an hour-based adaptive thresholding mechanism is chosen. Instead of applying a single global threshold across all time periods, the system maintains separate anomaly thresholds for each hour of the day and for each monitored feature. This design allows the detector to adapt to systematic variations such as daily traffic cycles and scheduled maintenance activities.

Thresholds are updated using the Exponential Moving Average (EMA), where the new threshold for hour h and feature i is computed as a weighted combination of the historical threshold and the 95th percentile of recent anomaly scores observed during the corresponding hour:

$$\theta_{h,i}^{\text{new}} = \alpha \cdot Q_{95}\left(E_{\text{combined}}^{(h,i)}\right) + (1-\alpha) \cdot \theta_{h,i}^{\text{old}} \tag{4.13}$$

where Q_{95} denotes the 95th percentile and $\alpha = 0.1$ controls the update rate. This approach ensures that thresholds adapt to changing patterns while remaining robust to temporary fluctuations. This approach was chosen for several reasons. First, network behaviour follow daily cycles, so static thresholds are insufficient. The hour-based approach creates 24 separate threshold profiles that account for these patterns. Secondly, the EMA method, with smoothing factor α equals to 0.1 allows thresholds to adapt gradually to new data without forgetting historical patterns. This balances stability, disregarding noise, while still capturing evolving patterns. Thirdly, mobile networks evolve over time. The exponential method gives weight to recent data while preserving accumulated knowledge, making it resilient to gradual changes in normal behaviour. Lastly, each feature e.g. latency, throughput, has independent thresholds, recognizing that different features have different variability patterns. Other methods that could be chosen are Max method, which is using the maximum value as threshold. This is disregarded due to being too sensitive to outliers and therefore making thresholds become unrealistically high [51]. Another method could be using a fixed percentile, however fixed percentile does not account for temporal patterns or concept drift [52].

4.1.2.2.6 Cross-Domain Correlation Analysis

After the anomaly detection, a cross-domain correlation analysis is applied to identify propagating faults. This is implemented through four sequential phases: temporal alignment, pattern clustering, statistical correlation, and insights generation.

4.1.2.2.6.1 Temporal Alignment Phase

Anomalies detected in different domains are first temporally aligned within a configurable time window with default 60 minutes to capture potential cascading relationships. The implementation uses a unidirectional forward scan where for each anomaly at time T_{source}, the algorithm identifies subsequent anomalies in other domains within $(T_{\text{source}}, T_{\text{source}} + \Delta t]$. This produces cross-domain event pairs:

$$\epsilon = (D_s, T_s, E_s) \rightarrow (D_t, T_t, E_t) \tag{4.14}$$

where D denotes domain, T timestamp, E anomaly error score, with $T_s < T_t$ and $|T_t - T_s| \leq \Delta t$. Large domains (>20,000 anomalies) are downsampled to 5,000 samples for computational feasibility.

4.1.2.2.6.2 Statistical Correlation Measurement

To quantify the dependency between anomaly patterns across domains, Spearman's rank correlation coefficient ρ is computed over aligned anomaly scores. Given two aligned anomaly sequences $\mathbf{X} = x_1, x_2, \ldots, x_n$ from domain D_1 and $\mathbf{Y} = y_1, y_2, \ldots, y_n$ from domain D_2, the correlation is calculated as:

$$\rho = 1 - \frac{6\sum_{i=1}^{n} d_i^2}{n(n^2-1)} \tag{4.15}$$

where $d_i = \text{rank}(x_i) - \text{rank}(y_i)$ represents the difference in ranks between paired anomalies and n is the number of aligned samples. The implementation includes optimization for large datasets through:

- Vectorized rank computation using PyTorch for GPU acceleration
- Intelligent sampling for datasets exceeding 1 million potential pairs, where intelligent sampling includes
  - size-aware sampling from the larger domain, to maximize coverage while minimizing comparisons, instead of arbitrarily sampling from either domain
  - prevents explosion by limiting pairs per timestamp

  - Before cross-domain analysis, domains with anomalies exceeding 20,000 are downsampled with a maximum of 5000 samples.
- Statistical significance testing with p-value computation using t-distribution approximation

Spearman's correlation is chosen due to its robustness to non-linear relationships compared to Pearson's correlation and its independence from absolute score magnitudes. By operating on rank orders rather than raw values, Spearman's correlation is less sensitive to outliers, skewed distributions and scale differences between variables [53].

4.1.2.2.6.3 Correlation Thresholding and Significance Analysis

The correlation analysis produces a symmetric matrix $\mathbf{C}$ where each element c_{ij} represents the correlation between domains D_i and D_j. Statistical significance is assessed at $\alpha = 0.05$ confidence level. Only correlations with $|\rho| > 0.3$ called "low positive correlation" [54] and $p < 0.05$ for statistical significance are retained for further analysis, filtering out spurious relationships. The system also tracks:

- Mean time difference between correlated anomalies
- Maximum anomaly score observed in correlated pairs
- Frequency of correlation occurrence across the analysis window

4.1.2.2.6.4 Pattern Discovery through Density-Based Clustering

Finally, correlated cross-domain anomaly events are clustered using HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise). HDBSCAN was chosen over e.g., DBSCAN or k-means, for several reasons:

- **Automatic cluster count**: Unlike k-means, HDBSCAN does not require pre-specifying the number of clusters, adapting to the structure of multi-domain failure patterns.
- **Noise identification**: HDBSCAN is usually more conservative and robust with noise than DBSCAN.
- **Varying density accommodation**: Different propagation patterns may exhibit different densities in the feature space, DBSCAN assumes uniform density and may encounter difficulties when clusters have different densities.

4.1.2.2.6.5 Two Stage Aggregation for Scalability

When processing thousands of cross-domain events, an aggregation approach is implemented:

1. **Event Aggregation**: Similar cross-domain events are aggregated within fixed time windows ((default: 15 minutes) based on source domain, target domain and time difference. This creates representative event groups to reduce dimensionality.
2. **Adaptive Clustering**: HDBSCAN is applied to aggregated events with minimum cluster size dynamically adjusted based on the total event count: max(2, total_events // aggregation_factor). This ensures appropriate cluster granularity regardless of data volume.

This two-stage approach enables efficient discovery of recurring multi-domain failure patterns while maintaining computational efficiency. The resulting clusters represent distinct cross-domain relationship patterns that can be analyzed.

4.1.2.2.6.6 Insights Generation The cross-domain analysis generates actionable insights by categorizing discovered patterns into:

- **Temporal Patterns**: Recurring sequences of domain failures with characteristic time delays, identified through event clustering
- **Correlation Patterns**: Statistically significant relationships between anomaly scores across domains using Spearman correlation

These insights are derived from the anomaly detection outputs and cross-domain event analysis. The correlation analysis identifies domain pairs with significant statistical relationships, while temporal patterns highlight recurring cross-domain event sequences.

4.1.2.2.7 Implementation Efficiency Considerations

The implementation of the LSTM autoencoder incorporates several optimizations to handle the computational demands of large data processing across multiple domains.

- **Batch processing**: Core domains (coreGx and coreRx) utilize batch sizes of 2048 samples, while E2E and RAN domains utilizes 4096 samples as the E2E and RAN data are larger. During training, the DataLoader implementation leverages PyTorch's memory pinning and multi-worker data loading to minimize I/O bottlenecks.
- **Validation split**: The validation split is 0.1, but is being reduced to 0.05 when there are more than 1 million samples. By reducing the validation, there is more training data, improving model performance, and lower validation cost, reducing memory overhead and training speed.
- **Memory Management**: Memory management system prevents memory exhaustion during extended processing sessions. First, Python garbage collection (gc.collect()) is used between domain processing stages. Second, CUDA memory clearing (torch.cuda.empty_cache()) is performed after each major computation phase. Lastly, adaptive batch sizing is applied by dynamically reducing the validation split from 10% to 5% once dataset size exceeds one million samples. This has multiple advantages, such as reducing computational overhead while preserving statistically sufficient validation and improving training stability during large scale runs.
- **Parallel Cross Domain Analysis**: Cross-domain analysis operations utilize concurrent execution through Python's ThreadPoolExecutor with configurable worker pools. Temporal alignment, correlation analysis and event clustering are distributed across multiple threads. This parallelization is particularly effective because cross-domain operations are largely I/O-bound, involving timestamp comparisons and file operations rather than matrix operations, which are more computational expensive.
- **Incremental Update Mechanism**: Rather than retraining models from scratch each day, autoencoder weights, optimizer states and learning rate scheduler states are preserved between training sessions, enabling continuation training with reduced epoch counts (from 75 initial epochs to 50 subsequent epochs). Threshold updates employ exponential smoothing with α equals 0.1, allowing gradual adaptation to evolving network patterns while maintaining stability against transient anomalies.
- **Threshold Building During Training**: During the training phase, the system exclusively builds statistical baselines of normal behaviour without performing anomaly detection. The hour-based thresholds are updated using exponential smoothing of the 95th percentile of reconstruction errors from training data. Therefore, no samples are labeled as anomalous and cross-domain analysis is not performed, ensuring the model learns normal behaviour patterns and minimizing computational overhead and overall training time. During testing, threshold updates are disabled. This ensures that the evaluation measures does not adapt on testing data.

4.1.2.2.8 Validation

The validation methodology implemented in this thesis is designed to analyze the behaviour of the LSTM autoencoder anomaly detection system. Given the absence of labeled anomaly data, the validation focuses on verifying model stability, convergence and behaviour by jointly analyzing raw KPIs, normalized input features and the binary anomaly indicators produced during the test phase of the model. For each validation date, the framework loads three aligned representations of the data:

- The original KPI measurements
- The normalized and windowed features used as input to the LSTM autoencoder
- The model output containing anomaly flags for each feature

The KPI's are directly recalculated from the raw counters in the original data. This recalculation is crucial for validation as it ensures that the data throughout all three datasets remain consistent. Furthermore, each domain configuration specifies distinct file paths, key identifier columns, feature mappings between original and normalized representations and rate calculation requirements.

The model stability and convergence are evaluated by computing anomaly rates across features, domains and days through aggregation. First, the feature anomaly rates of each feature are calculated by counting the binary anomaly flags relative to the total samples. This is performed for all features in the domain, resulting in individual anomaly percentages that indicate how frequently each specific metric triggers an alert. These feature level rates are stored in the feature_metrics dictionary. After computing individual feature rates, the results are aggregated to evaluate the domains. All anomaly instances and total samples across every feature within a domain is summed to compute a domain level anomaly rate. This aggregation is performed separately for each date in the evaluation period. The domain level results capture whether certain domains consistently produce more anomalies than others, which could indicate issues with specific domains. To assess model stability over time, cross date aggregation is performed. The average anomaly rates for each domain is computed across all evaluated dates, providing insight into whether anomaly detection patterns are consistent. Additionally, a global average anomaly rate is calculated by aggregating all anomalies and samples across every domain and date. This temporal aggregation enables detection of instability patterns, such as sudden spikes in anomaly rates on specific dates. For example, domain anomaly rates higher than 10% trigger warnings that the model's threshold may be overly sensitive, potentially generating excessive false positives. Conversely, rates below 1% prompt alerts that the model may be too conservative, potentially missing anomalies. These provide recommendations for threshold adjustment, feature normalization or model retraining.

For each feature within a domain, temporal indices of anomaly occurrences are identified and first-order differencing of these indices is computed to calculate anomaly time gaps between anomalies. Temporal clustering is performed by iterating through sorted anomaly indices and grouping consecutive anomalies where the gap does not exceed the configurable threshold, with default 6 indices, corresponding to 1.5 hours for 15 minute measurement intervals. The implementation uses a sequential scanning approach: starting from the first anomaly index, it accumulates indices into a current cluster until encountering a gap exceeding the threshold and begins a new cluster. This results into number of clusters and average cluster size statistics. Based on these metrics, detected anomalies are categorized into four pattern classes:

- **clustered events**: When at least one cluster exists and the average cluster size is at least 3, assigning a consistency score of 0.8.
- **small clusters**: When clusters exist but average size is less than 3, assigning a score of 0.6.
- **frequent scattered**: When no clusters exist but the average time gap between anomalies is less than 24 indices (6 hours), assigning a score of 0.4.
- **sparse scattered**: isolated, infrequent detections and are the default classification, assigned a score of 0.2.

In addition to pattern based validation, a simulated visual inspection procedure to approximate how a human operator would evaluate detected anomalies in practice. Z-scores express deviations in units of standard deviations and therefore provide a directly interpretable measure of how unusual a value is. This aligns with intuitive judgments of what appears anomalous to a human observer. Therefore, Z-score normalization was chosen. For each domain, 20% of all anomalies for each feature are used to maintain computational feasibility, the validation aligns anomaly indices with the corresponding raw KPI values and extracts a local temporal context spanning approximately 12 samples before and 12 samples after (3 hours for 15 minute intervals) each anomaly. Within this local window, the mean and standard deviation of the KPI values are computed, excluding the anomalous point, and a Z-score is calculated for the anomalous point with a small epsilon (e^{-10}) added to the denominator to prevent division by zero:

$$z = \frac{|x_{anomaly} - \mu_{context}|}{\sigma_{\text{context}}} \tag{4.16}$$

This Z-score indicates how strongly the anomalous value deviates from recent behaviour. Detected anomalies are then categorized according to their estimated visual distinction using threshold-based classification:

- **highly visible**: z > 3.0 with confidence score 0.9
- **moderately visible**: 2.0 < z ≤ 3.0 with confidence score 0.7

- **slightly visible**: $1.5 < z \leq 2.0$ with confidence score 0.5
- **not visible**: $z \leq 1.5$ with confidence score 0.3

Additionally, Trend break detection is evaluated by comparing the anomaly value with immediate neighbouring samples and checking whether the difference exceeds two standard deviations of the local context in both directions. The use of Z-scores in this step reflects the intuitive human perception of anomalies as values that are "significantly higher or lower than normal" within a recent temporal context, rather than as statistically outliers.

Finally, the validation results are aggregated across all evaluated dates and domains to produce an overall assessment of anomaly detection behaviour. Summary statistics include global and domain-specific average anomaly rates, pattern consistency distributions across the four pattern classes and the proportion of visually anomalies in the sampled inspection. The framework generates structured JSON reports containing raw metrics at feature, domain, dates and aggregated summaries. These are also visualized using that illustrate domain level differences via bar charts. The implementation also produces interpretative textual analysis with actionable recommendations. The complete implementation handles data loading from multiple sources (CSV and Parquet formats), performs necessary data transformations (rate calculations for certain KPIs), executes validation analyses and generates comprehensive outputs including JSON reports, visualizations and textual summaries and provides an evaluation for the LSTM anomaly detection, evaluating the stability and alignment with human observational reasoning due to the lack of available ground truths.

4.1.2.2.9 Model Persistence and Deployment

- **Model Checkpoints**: All critical system components are persistently stored. Autoencoder models have a checkpoint for each domain, including network weights. During incremental training sessions, optimizer states and learning rate scheduler configurations are maintained in memory to enable continued training across multiple days. Final trained models are saved as PyTorch state dictionaries for deployment.
- **Threshold Serialization Format**: Hour-based anomaly thresholds are serialized in JSON format. The serialization includes metadata such as: update method (exponential smoothing), smoothing factor (α=0.1) and percentile basis (95th).
- **Parallel Cross Domain Analysis**: Cross-domain analysis operations utilize concurrent execution through Python's ThreadPoolExecutor with configurable worker pools. Temporal alignment, correlation analysis and event clustering are distributed across multiple threads. This parallelization is particularly effective because cross-domain operations are largely I/O-bound, involving timestamp comparisons and file operations rather than matrix operations, which are more computational expensive.

4.1.2.3. Challenges

: Several practical challenges were addressed during the training process. This includes:

- **GPU**: Running deep learning models, especially LSTM autoencoders, on a CPU is extremely slow due to being sequential and computationally intensive. Training on a CPU significantly increases the total computation time and limits the ability to experiment with larger datasets or more complex models. Therefore, using a powerful GPU is essential, as GPUs can efficiently parallelize matrix operations and accelerate the training process.
- **Machine learning libraries**: PyTorch was chosen over TensorFlow for implementation. While TensorFlow provides extensive functionality, its latest versions currently does not support CUDA 12.8, which is the CUDA version of the laptop. In contrast, PyTorch offers better compatibility on Windows and supports a wider range of CUDA versions, making it more practical.
- **Anomaly threshold method**: The model implements an exponential moving average (EMA) hour-based thresholding method to detect anomalies from reconstruction errors. A major challenge with threshold-based anomaly detection is determining an appropriate threshold value. Unlike supervised classification, there is no clear notion of an optimal or best threshold method, as the threshold directly controls the trade off between sensitivity and stability. Small changes in

the threshold can significantly alter the number of detected anomalies and the absence of labeled anomalies further increases the difficulty to objectively tune this parameter. As a result, threshold selection relies on heuristics, domain knowledge, and empirical observation rather than a provably optimal solution.

- **Validation**: A fundamental challenge of this work is the absence of labeled ground truth anomaly data. As a result, classical supervised evaluation metrics such as accuracy, precision, recall, false positives or false negatives can not be computed. Validation therefore cannot be framed as measuring detection correctness, but rather as assessing whether the model behaves in a stable, consistent and interpretable manner. To address this, the validation focuses on indirect evidence by jointly analyzing raw KPI values, normalized model inputs and binary anomaly indicators produced by the LSTM autoencoder. Model stability is evaluated through aggregated anomaly rates across features, domains and dates, while temporal consistency is assessed using pattern based analysis of anomaly occurrences. In addition, a simulated visual inspection based on Z-score deviations within local temporal contexts is used to approximate how a human operator would perceive anomalies. While this multi-level validation framework provides insights into model convergence, sensitivity and alignment with intuitive human reasoning, the validation can not provide qualitative results on the behaviour of the model.

4.2. Offline Reinforcement Learning

The proposed root cause localization system is formulated as an offline reinforcement learning problem. The agent is trained on a fixed dataset of historical anomaly events, without any online interaction with the network environment. Its objective is to investigate a sequence of detected anomalies and for each anomaly, identify the past anomaly that triggered the root cause. This means the agent links each event to a past event that triggered it, building a causal chain backward in time. Conservative Q-Learning (CQL) is used to address the distribution shift that naturally arises in offline reinforcement learning. Since the model is trained exclusively on historical data without interacting with the environment, it may evaluate actions that were rarely observed or never encountered in the training dataset. In such cases, the model has insufficient evidence to accurately estimate their true value and traditional reinforcement learning algorithms can assign unrealistically high Q-values to these poorly represented actions, leading to overestimation and unstable policies. Quality values also known as Q-values represent the expected cumulative future reward of taking a specific action in a given state and thereafter continuing to act according to the current policy. When the model has little or no data about certain actions, these value estimates can become overly optimistic. CQL mitigates this issue by explicitly penalizing such value estimates and encouraging more conservative and reliable action selection.

4.2.1. Environment Formulation

The environment is modelled as a finite-horizon Markov Decision Process (MDP). An episode corresponds to a chronologically ordered list of anomaly events $e_0, e_1, \ldots, e_{T-1}$ from one or multiple network domains. At each time step t, the agent observes a state derived from the current event e_t and its recent history. It then selects an action that either declares a past event as the root cause of e_t or chooses to investigate further. The environment always moves to the next event e_{t+1} and returns a reward based on how likely it is to be the real cause of the selected action.

4.2.1.1. State Representation

The state is constructed from the current anomaly event and a fixed length window of events. Each event is encoded into a fixed dimensional feature vector that captures its domain identity, temporal characteristics, severity, and associated entity and feature identifiers. For an event e_i the extracted components are:

- **Domain one-hot encoding**: a binary vector of size D (number of domains) indicating the domain of the event. In this thesis, $D = 5$ (coreGx, coreRx, e2e_cell, e2e_pgw_sgw, ran_l).
- **Normalized timestamp**: $\tau_i = (t_i - t_0)/(t_{T-1} - t_0)$, where t_i is the absolute timestamp, scaled to $[0, 1]$ and T the total number of anomaly events in the episode.
- **Normalized error score**: $\hat{s}_i$, the anomaly score (reconstruction error) after robust IQR clipping and min–max scaling to $[0, 1]$.

- **Cyclical time features**: a 4-dimensional vector of the cyclical time features stated in Formula 4.3, 4.4, 4.5 and 4.6
- **Entity index**: an integer identifier that maps the entity information (e.g., cell id, eNodeB id, managed element) to a domain-specific vocabulary. The entity information is stored as a dictionary (e.g.,"managed_element": "A", "cell_id": "13"}). This dictionary is serialized into a JSON string with sorted keys to create a unique key. During training, each new key receives a new integer index starting from 1. Index 0 is reserved for out-of-vocabulary (OOV) entities that appear only during testing. *Example:* In domain `ran_1`, the entity might be `{"enodeb_id": "1", "local_cell_id": 3}`. During training this key gets index 1. If a new cell `{"enodeb_id": "ENB2", "local_cell_id": 5}` appears in testing, it maps to index 0.
- **Feature index**: an integer identifier for the specific monitored feature that triggered the anomaly with Index 0 denotes OOV, similarly mapped to a domain-specific vocabulary.

The encoding vector for each event is the concatenation of the components above. For a window of size L, the state s_t is formed by concatenating the encoding of the current event e_t with the encodings of the L preceding events $e_{t-1}, e_{t-2}, \ldots, e_{t-L}$. Events outside the episode boundary are padded with zero vectors. Therefore, the state always contains exactly $L+1$ event encodings. For example, with $L = 10$ and $t = 5$, the state includes e_5 and the ten preceding events $e_4, \ldots, e_0$, while indices < 0 are replaced by zero vectors. This representation yields a state dimension of $D{\cdot}(L{+}1){+}(L{+}1){\cdot}(1{+}1{+}4{+}2) = (L{+}1){\cdot}(D{+}8)$. The term $+2$ corresponds to the two integer indices (entity and feature).

4.2.1.2. Action Space Design

The action space is designed to capture the temporal nature of root cause propagation. At step t, the agent may select one of $L + 1$ discrete actions:

- **Action** 0: No root cause identified for the current event. This action incurs a small negative reward and does not terminate the episode.
- **Action** k ($1 \leq k \leq L$): The event at index $t - k$ (i.e., k steps in the past) is declared the root cause of e_t.

This design lets the agent link an anomaly to any event within the window, helping it learn how effects spread over time and how long delays occur.

4.2.1.3. Reward Function

The reward function is designed to reflect both the candidate root cause and the statistical and temporal evidence supporting the causal link. For an action $k > 0$ at time t with root candidate $r = e_{t-k}$, the reward R_t is computed as:

$$R_t = 0.5 \cdot \hat{s}_r + \sum_{\substack{j>t \\ t_j \leq t_r + \Delta_{\text{future}}}} w_{\text{time}}(t_j - t_r) \cdot p_{\text{delay}}(d_r, d_j) \cdot \hat{s}_j \tag{4.17}$$

where:

- $\hat{s}_r$, $\hat{s}_j$ are normalized error scores of root and subsequent events.
- $\Delta_{\text{future}} = 120$ minutes defines the future evidence window starting from the root event's timestamp.
- $w_{\text{time}}(\delta) = 1/(1 + \delta)$ is a time-decay weighting factor. The time-decay weighting ensures that anomalies closer in time to the root contribute more, reflecting the higher likelihood of a direct causal link.
- $p_{\text{delay}}(d_1, d_2)$ is a plausibility score based on historical cross-domain delay statistics. It is defined as $\tanh(\text{count}(d_1 \to d_2)/10)$, where count is the number of observed (ordered) domain pairs within a maximum delay of 120 minutes in the training data.

Additional reward shaping terms are applied:

- If the time difference $\delta = t_r - t_t$ is negative (root appears after target) and $|\delta| > \delta_{\text{simult}} = 7.5$ minutes, a penalty of -0.5 is incurred.

- If $\delta < 0$ but within the simultaneous window, the root severity contribution is reduced by a factor 0.8.
- If $\delta > 60$ minutes, a penalty linearly increasing with δ is subtracted.
- If the root and target are the same event (self-cause), the specific rule for that domain is checked. If self-cause is not allowed for that domain, the action receives a penalty of -0.3.
- If the agent selects action 0, a fixed penalty $R_t = -0.2$ is returned.

4.2.2. GPU-Optimized Q-Network with Domain-Specific Embeddings

The Q-function is approximated by a deep neural network that leverages GPU parallelism. A critical challenge is handling categorical entity and feature identifiers whose vocabularies vary across domains and can be large. To address this, the network uses embedding tables for each domain and a vectorized lookup procedure that avoids Python loops.

4.2.2.1. Architecture

The network receives the flattened state vector and first reshapes it to a batch of per-event vectors of length $(L + 1)$. The per-event components are then split:

- Domain one-hot encoding: used to derive the domain ID via arg max.
- Time norm, error norm, and cyclical time features: passed directly as continuous features.
- Entity index and feature index: used to index into embedding tables.

For each domain, an embedding table is maintained for entities and features, each with a reserved index 0 for OOV entries. The forward pass performs lookups in batches: a mask is created for positions belonging to a given domain, the corresponding indices are clamped to the vocabulary size and the embeddings are fetched in one operation. The embeddings are then scattered back into the output tensor using masked_scatter. This vectorized implementation eliminates per event or per domain loops and accelerates training on GPU.

For example: assume there are two domains: coreGx with entity vocabulary size 101 (index 0 for OOV + 100 known entities) and feature vocabulary size 51. ran_l with entity vocabulary size 201 and feature vocabulary size 31. In a batch of 256 sequences, each of length 31 (L + 1), we have tensors of domain IDs (shape [256,31]) and entity indices (shape [256,31]). Instead of looping over each of the $256 \cdot 31$ positions and branching by domain, a boolean mask for coreGx: mask_core = (domain_ids == coreGx_id). Then, the entity indices where mask_core is True is extracted giving idx_core = entity_idx[mask_core]. idx_core is clamped to the valid range for coreGx: [0, 100] (ensuring OOV index 0 stays 0). Single batched lookup is performed: emb_core = coreGx_entity_embedding(idx_core). Finally, these embeddings are scattered back into the output tensor and the same steps are repeated for ran_l.

The continuous features for each event, one-hot domain encoding, and the retrieved entity/feature embeddings are concatenated to form a unified per-event representation of dimension $D + 1 + 1 + 4 + E_{\text{ent}} + E_{\text{feat}}$, where $E_{\text{ent}} = 16$ and $E_{\text{feat}} = 8$ are the chosen embedding dimensions. Entity embeddings need to capture more complex information because entities are numerous and varied. A larger dimension allows more variety of representations. Feature embeddings are smaller because feature names are fewer and less diverse. The full sequence of per-event vectors (for the current event and the L past events) is flattened into a single long vector. This vector is then passed through a two-layer MLP with hidden dimension 256 and ReLU activations, followed by a linear output layer of size $L + 1$. Each output corresponds to the estimated Q-value of one action: action 0 (no root cause) and actions 1 through L (which point to the event k steps in the past). Thus the network directly predicts the expected future reward for each possible root cause declaration given the current state.

To clarify this, consider a lookback window $L = 2$, so the state contains three events: the current event e_t and the two preceding events e_{t-1}, e_{t-2}. Suppose there are $D = 2$ domains: “core” (index 0) and “ran” (index 1). For simplicity, assume a batch size of 1. The flattened state vector is reshaped into a $3 \times (D + 8)$ matrix, i.e., 3×10, where each row corresponds to an event and contains:

- domain one-hot (2 numbers),

- normalized timestamp (1),
- normalized error score (1),
- cyclical time features (4),
- entity index (1 integer),
- feature index (1 integer).

Let the current event e_t be the domain "core", so its one-hot vector is $[1, 0]$. Its entity index is, e.g. 5 and its feature index is 3. From the one-hot we derive the domain ID 0 (core) via arg max. The entity index 5 is used to look up an embedding from the core-specific entity table. If the core entity vocabulary size is 100, index 5 is valid and returns a 16-dimensional vector. (If the index had been out of range, it would be clamped to the nearest valid index) Similarly, feature index 3 looks up an 8-dimensional embedding from the core feature table. The same process is repeated for the two past events, which may belong to other domains (e.g., e_{t-1} could be "ran") and use their respective embedding tables. After concatenation, each event yields a per-event vector of dimension $2 + 1 + 1 + 4 + 16 + 8 = 32$. The three such vectors are flattened into a single vector of length $3 \times 32 = 96$, which is then fed to the MLP, MLP (Multi-Layer Perceptron), which is a feedforward neural network composed of stacked fully connected layers, to produce Q-values for the $L + 1 = 3$ possible actions:

- Action 0: No root cause identified for the current event.
- Action 1: The event at index $t - 1$ (1 step in the past) is declared the root cause.
- Action 2: The event at index $t - 2$ (2 steps in the past) is declared the root cause.

4.2.2.2. Conservative Q-Learning (CQL)

To mitigate the overestimation bias common in offline RL, the Conservative Q-Learning algorithm is chosen. The loss function augments the standard Bellman error with a regularization term that minimizes Q-values on out-of-distribution actions, actions that do not appear in the offline dataset and may therefore be unreliable, while maximizing Q-values on actions observed in the dataset. The Bellman error measures the discrepancy between the current Q-value estimate and the target value specified by the Bellman equation and is defined in Equation 4.18.

$$\text{Bellman error} = \big(Q(s,a) - (r + \gamma \max_{a'} Q_{\text{target}}(s', a'))\big)^2, \tag{4.18}$$

where $Q(s, a)$ is the current estimate, Q_{target} is a target network (a slowly updated copy) used for stability, r is the reward, γ is the discount factor and s' is the next state.

CQL incorporates an additional regularization term that penalizes Q-values for out-of-distribution actions, while rewarding Q-values for actions observed in the dataset. This encourages the model to remain conservative, assigning high value only to supported actions and reducing overestimating Q-values for unseen or unlikely actions. The resulting total loss is:

$$L = L_{\text{TD}} + \alpha \cdot L_{\text{CQL}}, \tag{4.19}$$

where the temporal difference (TD) loss is the expectation of this Bellman error over transitions sampled from the dataset:

$$L_{\text{TD}} = \mathbb{E}_{(s,a,r,s') \sim D}\Big[\big(Q(s,a) - \big(r + \gamma \max_{a'} Q_{\text{target}}(s', a')\big)\big)^2\Big], \tag{4.20}$$

where s is the history of anomaly events with embeddings for entities and features, a is the selected root cause offset (0 meaning no root cause) and r is a reward computed based on temporal plausibility, severity and downstream evidence. The TD loss ensures that the Q-function learns to predict the expected future reward for each candidate root cause decision.

and the CQL loss is:

$$L_{\text{CQL}} = \mathbb{E}_{s \sim D}\left[\log \sum_a \exp(Q(s,a)) - \mathbb{E}_{a \sim D}\big[Q(s,a)\big]\right]. \tag{4.21}$$

where the first term, is the log-sum-exp over all possible actions in state s. This quantity approximates the maximum Q-value and penalizes high Q-values for any action, especially those that are not grounded in the data. The second term is the average Q-value of the actions that actually appear in the dataset for state s. This term encourages the Q-values of in-distribution actions to be high.

L_{CQL} acts as a regularizer that pushes the Q-function to assign relatively higher values to actions seen in the data and lower values to all actions. The net effect is that the policy remains conservative. High confidence is only assigned to actions that are supported by the dataset, reducing the risk of overestimating the value of rare or unseen root cause choices.

The hyperparameters are set to

- $\gamma = 0.99$: Controls how much the agent values future rewards.
- $\alpha = 2.0$: how strongly the conservative penalty L_{CQL} influences training.
- learning rate $= 3 \times 10^{-4}$: Determines how large each gradient update step is.
- target network update frequency of 100 steps.

4.2.3. Offline Dataset Generation

The offline dataset is generated by simulating a mixed behaviour policy on the training event sequences. For each event index $t \geq L$, a transition (s_t, a_t, r_t, s_{t+1}) is created. The action is sampled from a mixture of three strategies:

- *Random* (40%): uniformly choose any action in $0, \ldots, L$.
- *Strongest anomaly* (30%): select the past event within the window with the highest error score.
- *Earliest within 60 minutes* (30%): select the most recent past event that occurred within 60 minutes prior to e_t.

This mixture tries to cover both conservative and aggressive root cause attributions.

The generated transitions are stored as NumPy arrays and subsequently transferred to GPU memory as contiguous tensors. A PyTorch DataLoader with multiple workers loads mini batches directly from GPU memory, eliminating CPU to GPU transfer overhead during training.

4.2.4. Evaluation Protocol

Root cause assignments produced by the trained policy are validated using a statistical significance test and rules for each domain.

4.2.4.1. Validation of a Root Cause Claim

A root cause attribution $r \to t$ is considered *valid* if it satisfies all of the following criteria:

1. **Temporal plausibility**: The time difference $t_t - t_r$ must not be less than $-\delta_{\text{simult}}$ (i.e., root must not appear significantly after the target). With other words, the root can occur at most δ_{simult} minutes later than the target, this allows for cases where the root and target are essentially simultaneous. This is set at 7.5 minutes.
2. **Self-cause policy**: If $r = t$, the domain must explicitly allow self-causes, and there must be at least a minimum number of subsequent anomalies in the future window. In root cause analysis, labeling an event as its own cause is meaningful only if that event triggers further anomalies. Without downstream evidence, a self-cause claim is saying "this event caused itself", which is neither informative or verifiable. The policy ensures that self-causes are only considered when they are the starting point of a propagation chain.
3. **Chain length**: At least $C_{\min} = 3$ distinct anomalies must lie between r and t. The focus is on propagation chains rather than isolated anomaly. In real networks, a root cause typically triggers a cascade of anomalies across multiple components. An isolated anomaly might be coincidental or due to a common external factor. By requiring a minimum chain length, we ensure that the attribution is supported by multiple pieces of evidence, increasing the confidence that a genuine causal relationship exists. This does not mean that isolated spikes are irrelevant, but in the

evaluation, this will not counted as valid root causes unless they are part of a longer chain. This is a design choice to focus on propagations.

4. **Statistical significance**: A two-sample Kolmogorov–Smirnov test is performed to compare the anomaly scores of events in the causal chain against a background window of 4 hours surrounding the chain. The resulting p-value must be less than 0.05.

   For example, consider a chain with anomaly scores $[0.85, 0.92, 0.78, 0.88]$ and a background set of scores $[0.32, 0.45, 0.51, 0.29, 0.63, 0.41, 0.37, 0.55]$ from the surrounding four hours (2 before the earliest chain event and 2 hours after the latest chain event). The KS test compares the empirical cumulative distributions of these two samples. A low p-value (< 0.05) indicates that the chain's scores are statistically unlikely to come from the same distribution as the background, supporting that the chain represents an anomaly pattern. The KS test is chosen because it is non-parametric (makes no assumptions about normality), sensitive to any difference in distribution (location, spread, or shape) and works well with the small sample sizes typical of causal chains.

A confidence score is computed as:

$$\text{confidence} = \min\Big(1.0,\ 0.3 + 0.1 \cdot |\text{chain}| + 0.2 \cdot (1 - p) + 0.2 \cdot \hat{s}_r\Big), \tag{4.22}$$

where $|\text{chain}|$ is the number of distinct anomalies in the chain (root and target inclusive), p is the p-value obtained from the two-sample Kolmogorov–Smirnov test ($1 - p$ increases with statistical significance) and $\hat{s}_r$ is the normalized anomaly score of the root event (scaled to $[0, 1]$).

4.2.4.2. Metrics

The primary metric is the success rate, defined as the fraction of assigned events that are validated. Secondary metrics include the percentage of events assigned, self-cause rate and distribution of root cause domains. For comparison, a baseline heuristic is evaluated: earliest within 15 minutes, which selects the most recent past anomaly within a 15-minute window, or the strongest anomaly if none exists.

4.2.4.3. Handling of unknown entities and features.

During training, the environment maps any unseen entity or feature to the reserved index 0 (OOV). Moreover, when the state window extends beyond the episode start, the missing events are padded with zero vectors; these padding positions also use index 0 for entity and feature. Consequently, the network learns an embedding for the OOV token from both genuine unknown events and padded slots. This learned representation for "unknown" is then reused during testing when genuinely new entities or features appear, ensuring that the network never encounters an out-of-range index. The clamping inside the forward pass is only an additional safety measure and does not normally alter the indices. Moreover, the same cross-domain delay statistics computed from training data are used during testing, reflecting a realistic scenario where future causal patterns are used from historical observations.

4.2.4.4. Limitations

First, the model is constrained to assign root causes only to past anomalies present in the observed event stream. It cannot attribute an anomaly to non-anomalous events, that are not captured by the anomaly detection model. Furthermore, the cross-domain statistics used in both the reward function and the evaluation protocol are derived from a sampled subset of anomaly events. While this sampling is essential computationally, it may result in an incomplete picture of the true delay distributions. The RL agent therefore learns from a potentially biased sample, which could cause it to overlook cross-domain propagation patterns. Furthermore, the statistical validation of root cause chains also relies on these same sampled statistics. This creates a trade-off between scalability and representativeness. This is an intrinsic limitation of any approach that relies solely on anomaly event data. True root causes that are not detected anomalies remain inaccessible to the model. Furthermore, the anomaly detection model produces spikes at the beginning of an anomaly and at the end of an anomaly. During the period of consistent high anomaly, it may not flag the whole period as anomaly due to the behaviour of the autoencoder. This behaviour can affect the accuracy of root cause identification, as the RL model relies on these detected anomaly events to assign root causes.

Second, the offline dataset is generated using a heuristic behaviour, a mixture of random selection, strongest anomaly within the window and the earliest anomaly within 60 minutes. The reward function is designed to counteract the sub optimality of these heuristics and to encourage discovery of statistically causal chains. The absence of labeled root cause assignments remains a significant limitation. In an ideal setting, the dataset would contain traces of confirmed root cause labels. However, in the unsupervised context typical of real world network operations where such labels are unavailable, this heuristic generation could be a defensible choice that enables offline RL training.

Third, the evaluation also relies on heuristic validation criteria rather than ground truth. The combination of temporal plausibility, rules for each domain, chain length constraints and statistical significance testing provides a way for validation, but it does not constitute definitive proof of causation. The validation function may be biased toward chains that exhibit strong statistical patterns, e.g., unusually high anomaly scores, even when those patterns could arise from coincidental correlations rather than true causal propagation. On the other hand, genuine causal relationships that appear as subtle or may be incorrectly rejected. The challenge of the absence of ground truth for root cause in complex, multi-domain systems is a difficulty in unsupervised root cause analysis and is shared by practically all methods operating in this domain. The validation approach represents an effort to ensure robustness and practical applicability, but the results should still be interpreted as suggestions rather than definitive conclusions.

In this problem, the offline dataset is collected from historical anomaly logs. It is impossible to guarantee coverage of every possible root cause chain, especially cross-domain propagations. CQL helps the model avoid overconfident predictions on unfamiliar patterns, improving the reliability of the identified root causes.

5

Results

This chapter presents and analyzes the results obtained from the LSTM autoencoder and the reinforcement learning model described in Chapter 4. The objective of these experiments is to evaluate the ability of the proposed models to learn normal system behaviour and to identify deviations indicative of anomalies across different network domains.

5.1. Anomaly detection

Anomaly detection is performed using an LSTM-based autoencoder trained on normal operational data from each domain. The model learns to reconstruct how normal data behave, where deviations from the learned patterns are reflected in an increased reconstruction error. The reconstruction error is defined as the difference between the original input time series and its reconstructed output produced by the autoencoder.

The training and validation loss curves for each domain are shown in Figures 5.1, 5.2, 5.3, 5.4 and 5.5 respectively. The training dates spans from 26 August 2025 till 12 September of 2025 and 17 September 2025 till 30 September 2025. In all cases, the loss decreases rapidly during the initial training epochs, followed by a gradual convergence, indicating that the model is learning the underlying temporal patterns present in the data. A close alignment between training and validation loss can be observed across all domains, with the training loss slightly higher than the validation loss in most cases. This behaviour can be observed due to dropout being applied during training, as they increase the training loss while being disabled during validation. The small difference indicates that the models generalize well and that overfitting is limited. Differences in convergence speed and initial losses values are visible between domains. The latter vary drastically between domains, e.g., Core and RAN starts below 1, E2E cell starts around 350 and E2E PGW SGW starts around 7. This is a strong indicator of the scale and variance of the input features. Furthermore, several of the domains show distinct spikes in the loss curves, these could indicate that the model is encountering batches containing less common patterns, transitions of weekday/weekend patterns or temporal dynamics to each domain. Smaller data or domains that are more stable and predictable behaviour converge faster and consequently obtain a lower reconstruction error, whereas domains characterized by higher variability require longer training to reach comparable performance. The stabilization of the reconstruction loss towards the end of training suggests that the learned representations are robust and applicable to unseen normal data. This is critical for anomaly detection, as it ensures that increases in reconstruction error during inference can be reliably associated with abnormal system behaviour rather than model instability or insufficient generalization. These training results positively indicate strong generalization, resilience for spikes and that the LSTM autoencoders are capable of modeling normal behaviour for each domain.

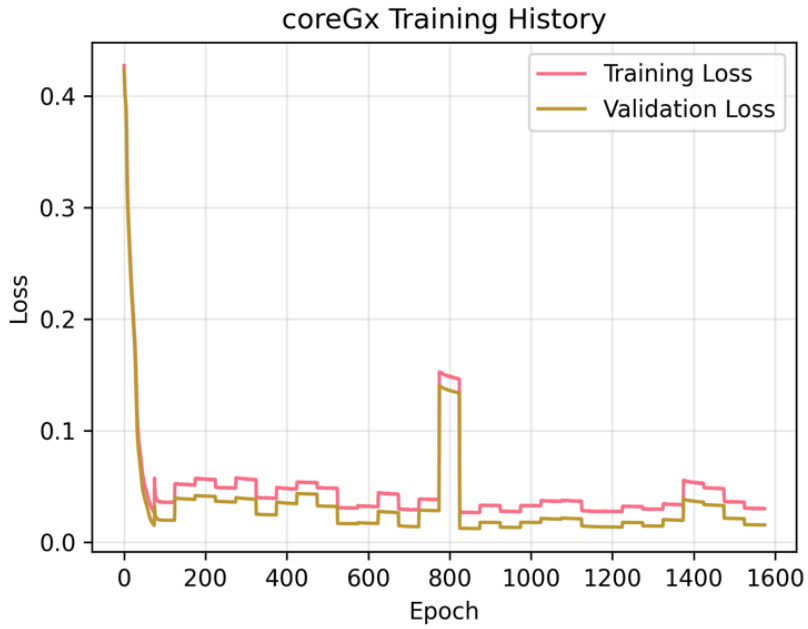

Figure 5.1: The reconstruction loss of the LSTM training of CoreGx

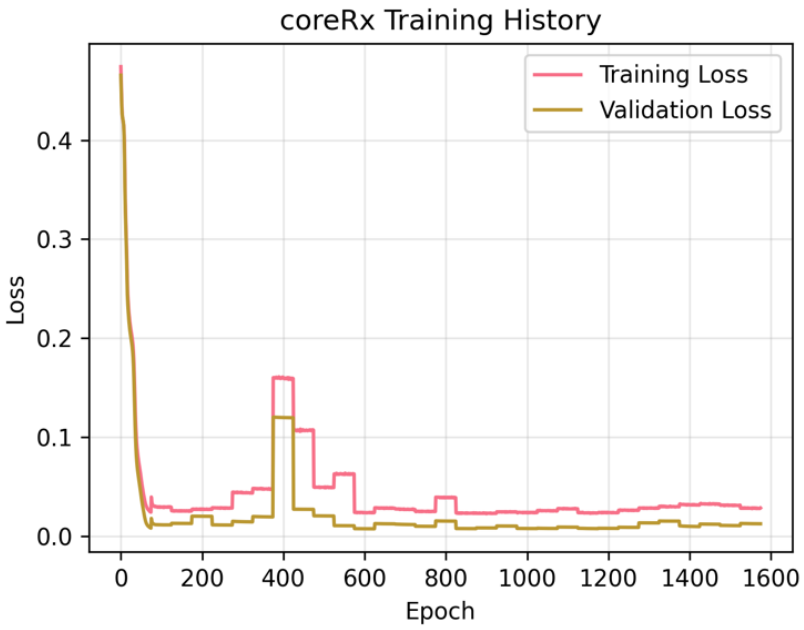

Figure 5.2: The reconstruction loss of the LSTM training of CoreRx

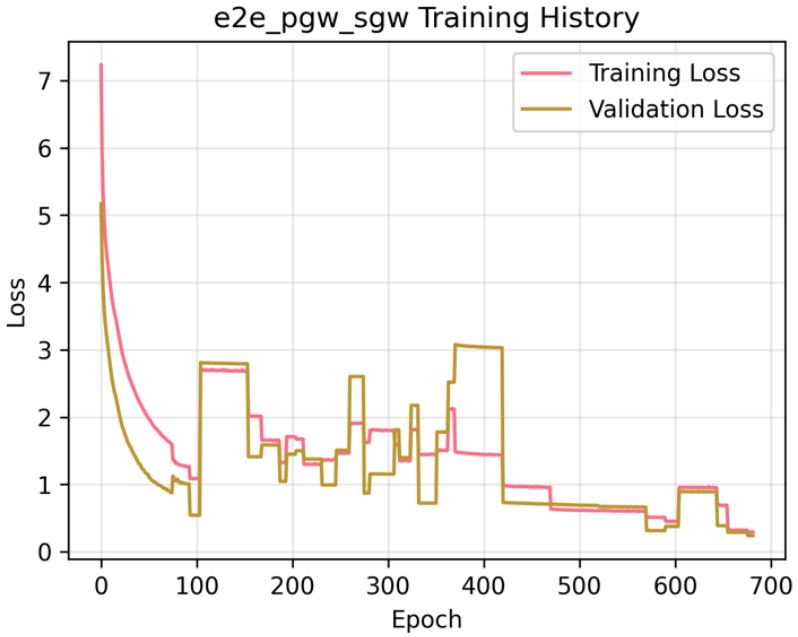

Figure 5.3: The reconstruction loss of the LSTM training of E2E PGW SGW

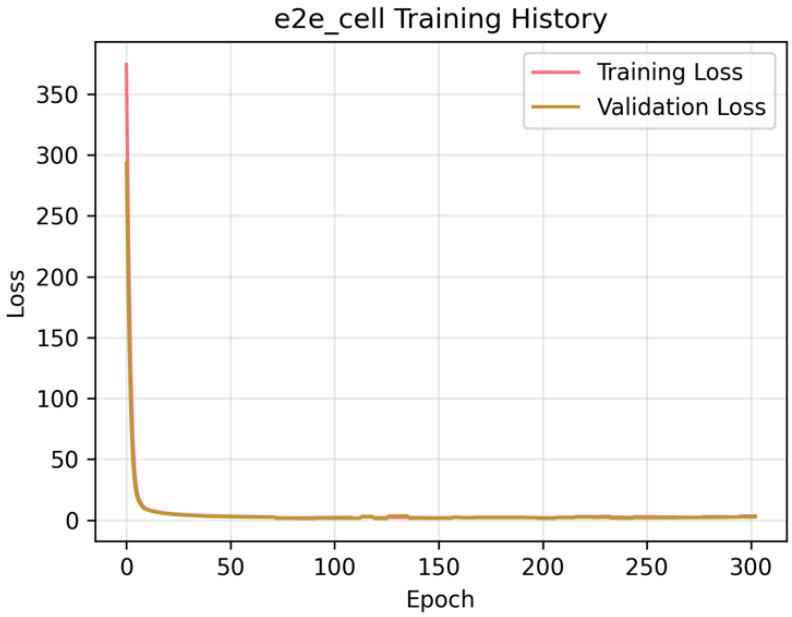

Figure 5.4: The reconstruction loss of the LSTM training of E2E Cell

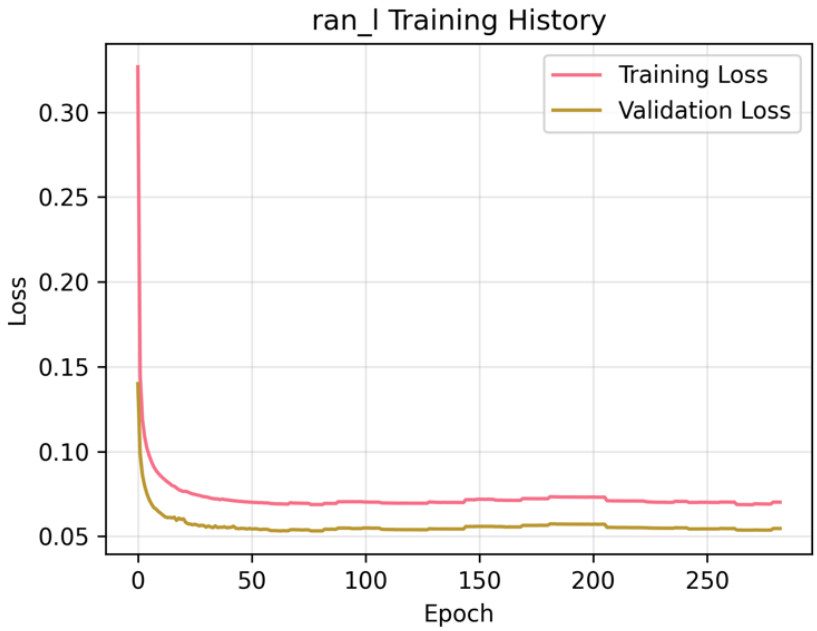

Figure 5.5: The reconstruction loss of the LSTM training of RAN

Figure 5.6 presents the RRC reestablishment success rate for the selected eNodeB id 8421 and local cell 13 across all training dates together with the test days of 22, 23, 24 August 2025 and 1 October 2025 of the original KPI. The 24th of August is highlighted as a clear anomaly can be seen and will be compared with the anomaly detection model. As shown in the figure, the raw measurements shows substantial noise, particularly during night time hours. This behaviour is expected, since between 00:00 and 07:00 the number of active users tends to be extremely low. In such conditions, even a single failed reestablishment attempt can lead to large fluctuations in the success rate percentage, e.g., one failure out of two attempts already implies a 50% success rate. For this reason, excluding these low traffic hours from model training and evaluation may improve the stability of the input data.

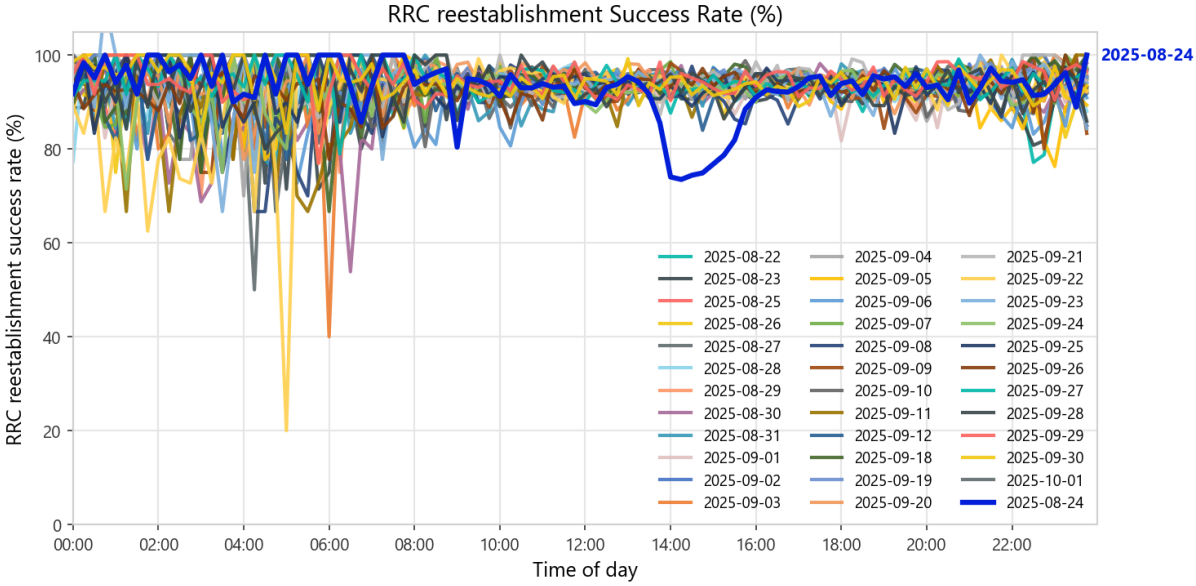

Figure 5.6: Original data of *RRC_reestablish_success_rate*

Figure 5.7 shows the test day of 24 August 2025, with the original KPI values plotted in the top plot

and the model output with on the y-axis the corresponding model combined error in the bottom plot with the same time axis. The combined error is the error shown in Formula 4.12 which is a combination of the three error components. The model clearly detects the anomalous behaviour occurring around 15:00, where the combined error exceeds the learned threshold, in the figure called anomaly threshold. However, several additional spikes in the combined error appear throughout the day that are above the anomaly threshold, but appears normal in the training dates. This suggests that the threshold may be set too low, resulting in an excessive number of points being flagged as anomalous. By adjusting the threshold for example optimizing against a validation set could help reduce false positives while maintaining sensitivity to real anomalies.

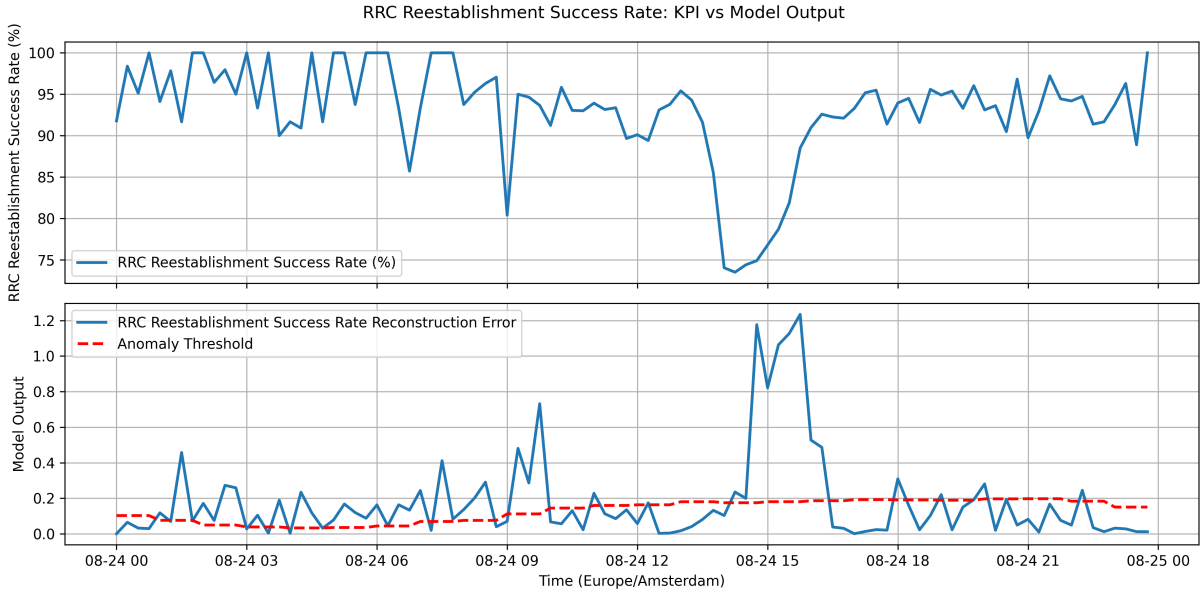

Figure 5.7: Comparison of RRC reestablishment success rate (top) and model output (bottom) over time

Similarly, in Figure 5.8, the PRB downlink utilization for eNodeB 34001 and local cell id 12 across all training and test dates of the original KPI can be seen. The test date that is highlighted here is 1 October 2025, where a clear anomaly can be seen during the end of the day. In this figure the noise during night time hours is less visible which is reasonable since the PRB downlink utilization is an aggregated and continuous load metric that is less sensitive to low traffic volumes. As a result, even when there are few active users the utilization remains close to zero and is therefore strongly attenuated by individual transmission or event driven events or short term fluctuations.

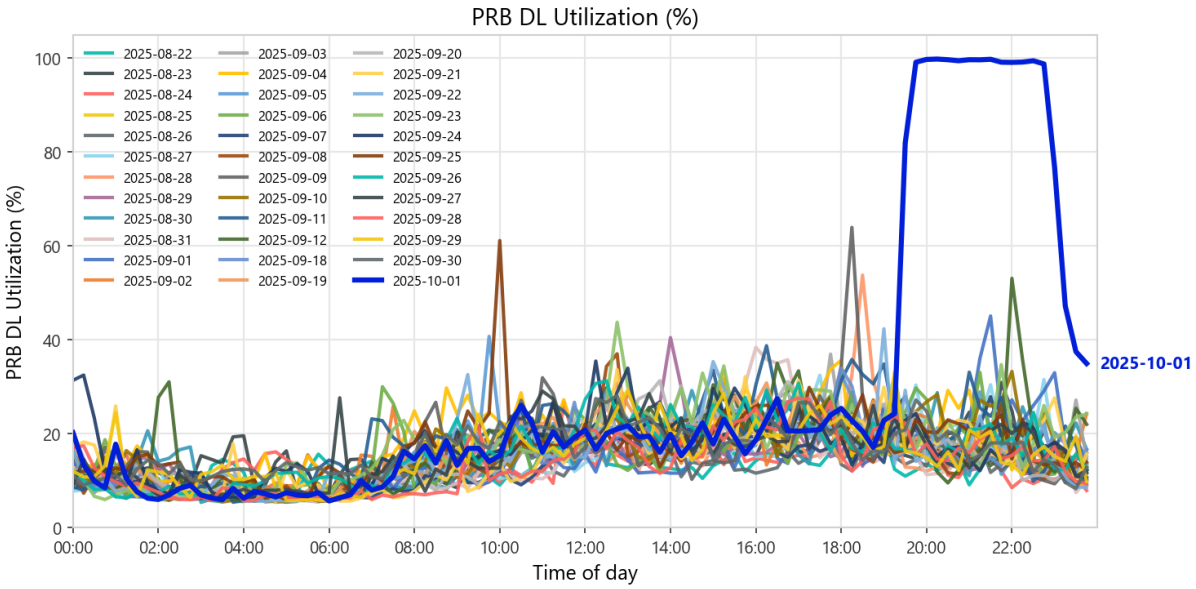

Figure 5.8: Original data of *PRB_DL_utilization*

Figure 5.9 shows all test days. The top plot shows the original KPI values and the bottom plot shows the combined error with the corresponding model combined error with the same time axis. The model appears to detect the start of the anomaly, but only as a sharp spike, while in the top plot the anomaly seems to persist for a few hours, while the combined error only shows relative small peaks. When the PRB utilization drops, a relatively high spike in the model combined error reappears. Although the PRB DL utilization reaches nearly 100% for several consecutive hours, the anomaly detection model produces only short spikes in its combined error rather than a continuously relatively high anomaly score. This behaviour is typical of autoencoder-based detections, which are primarily sensitive to deviations in temporal structure rather than absolute signal magnitude. The transition from normal utilization to unexpected high utilization produces a large combined error. Once the high utilization stabilizes into a new, nearly constant pattern, it no longer violates the learned temporal relationships and becomes easier for the autoencoder to reconstruct, despite being anomalous in value. As a result, the model only shows a spike at the start of anomaly, but not the continuous anomalous period itself. This can also explain the second spike appearing at the end of the high utilization period. Lastly, three additional test dates are plotted in the same graph to emphasize that the spike of the model is not influenced by the other test dates, but belongs to the actual date, which is the only date with an anomaly.

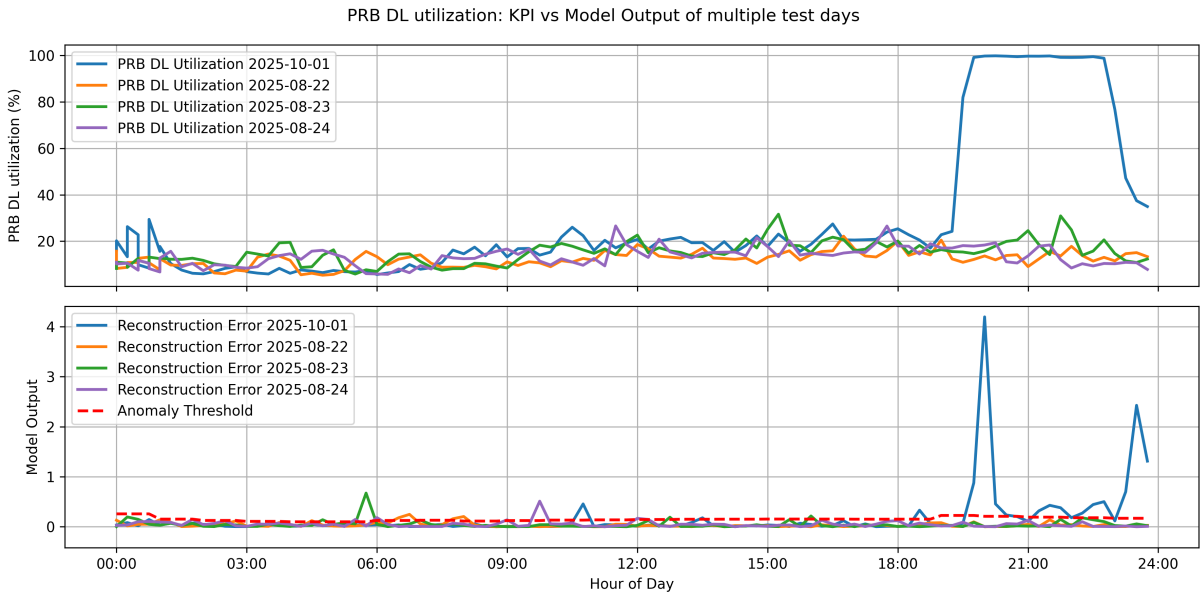

Figure 5.9: Comparison of PRB DL utilization (top) and model output (bottom) over time

In Figure 5.10, the lower plot is more zoomed in with the y-axis limited to 0.5 in order to see the anomaly threshold line. Here a lot of fluctuations around the threshold line can be seen. The zoomed in view shows some fluctuations around the anomaly threshold even when it appears normal in Figure 5.9. Several normal test days produce peaks that either reach or nearly reach the threshold, which could indicate potential false positive as it appears normal during the training.
This suggests that the threshold may be set too low, resulting in an excessive number of points being flagged as anomalous. By adjusting the threshold for example optimizing against a validation set could help reduce false positives while maintaining sensitivity to real anomalies. Several normal test days produce peaks that either reach or nearly reach the threshold, indicating potential false positive sensitivity. The anomaly on 1 October 2025 produces combined error magnitudes higher than the threshold, suggesting that a higher threshold would still reliably flag the anomaly while reducing false alarms. This indicates that the current threshold is overly conservative and could be shifted upward to improve robustness.

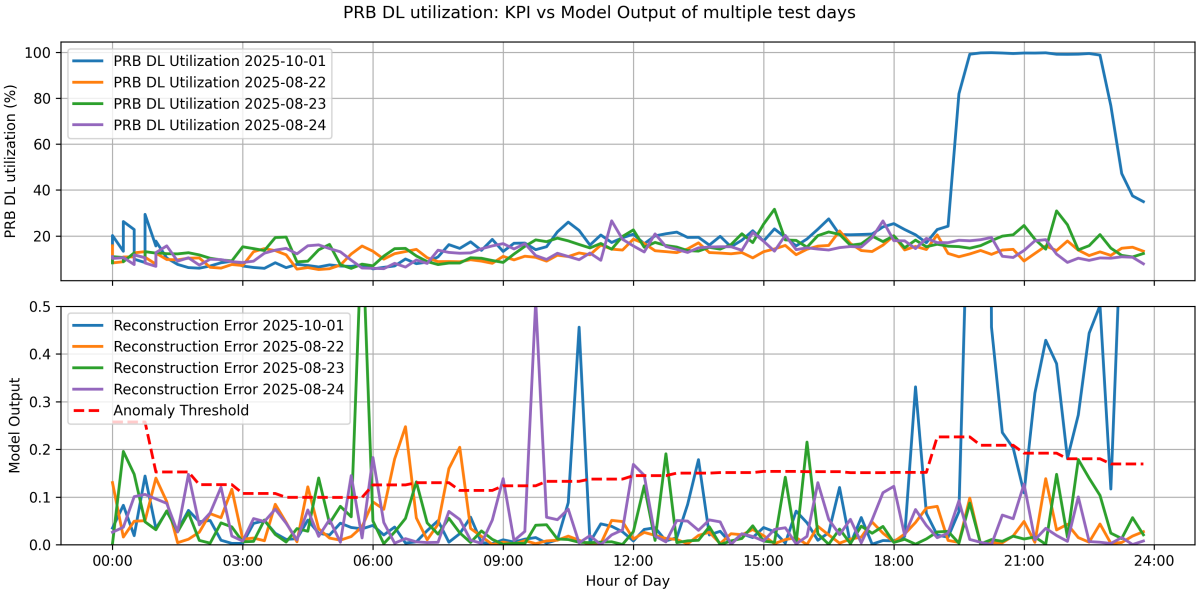

Figure 5.10: Comparison of PRB DL utilization (top) and model output (bottom) over time zoomed in

Figure 5.11 presents the Out of Order Packets Downlink rate for the selected cell id 8307479 across all training dates together with the test days of 22, 23, 24 August 2025 and 1 October 2025 of the original KPI. The 23th of August is highlighted as a clear anomaly can be seen and will be compared with the anomaly detection model. In this figure, the noisy behaviour during night time is also visible.

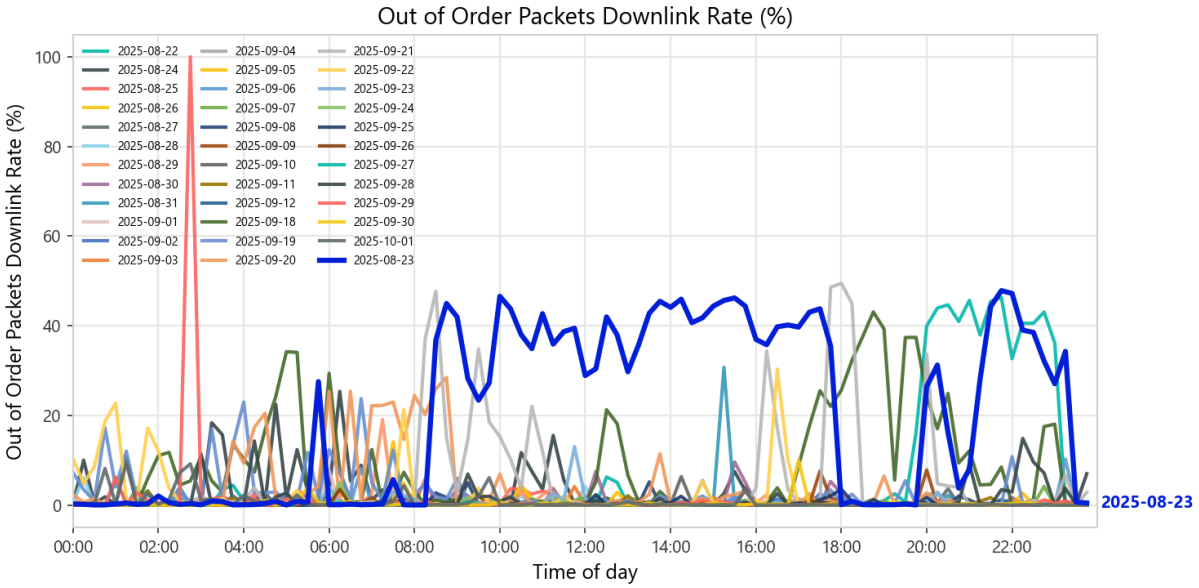

Figure 5.11: Original data of *Out_of_order_packets_downlink*

Figure 5.12 shows on the top plot the original data of the highlighted date, 23 August 2025, and bottom plot the output of the anomaly model. An anomaly occurring around 06:00 is clearly identified by the model, so is the anomaly around 08:30, which occurs as a spike in the model output. Although the anomaly persisting around 18:00 shows fluctuations in the magnitude, the combined error decreases shortly after its rise while remaining above the anomaly threshold. When the anomaly drops around 18:00, another significant spike is observed. A similar pattern appears for the anomaly starting around 21:00 lasting until the end of the day, resulting in another increase in the combined error. Overall, the

model successfully detects all anomalies present on the selected date.

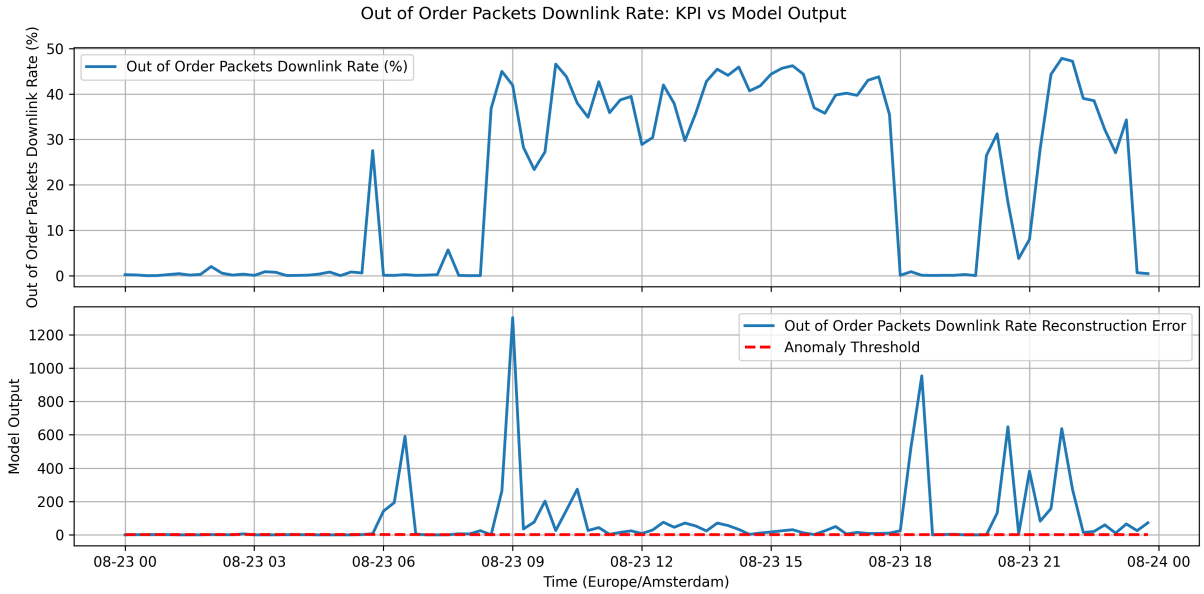

Figure 5.12: Comparison of Out of order packets downlink (top) and model output (bottom) over time

Figure 5.13 shows a zoomed in view of the model output. There is a peak occurring around 03:00 which is not visible in the original data. In addition, the persistent anomaly observed from approximately 08:30 to 18:00 consistently exceeds the anomaly threshold.

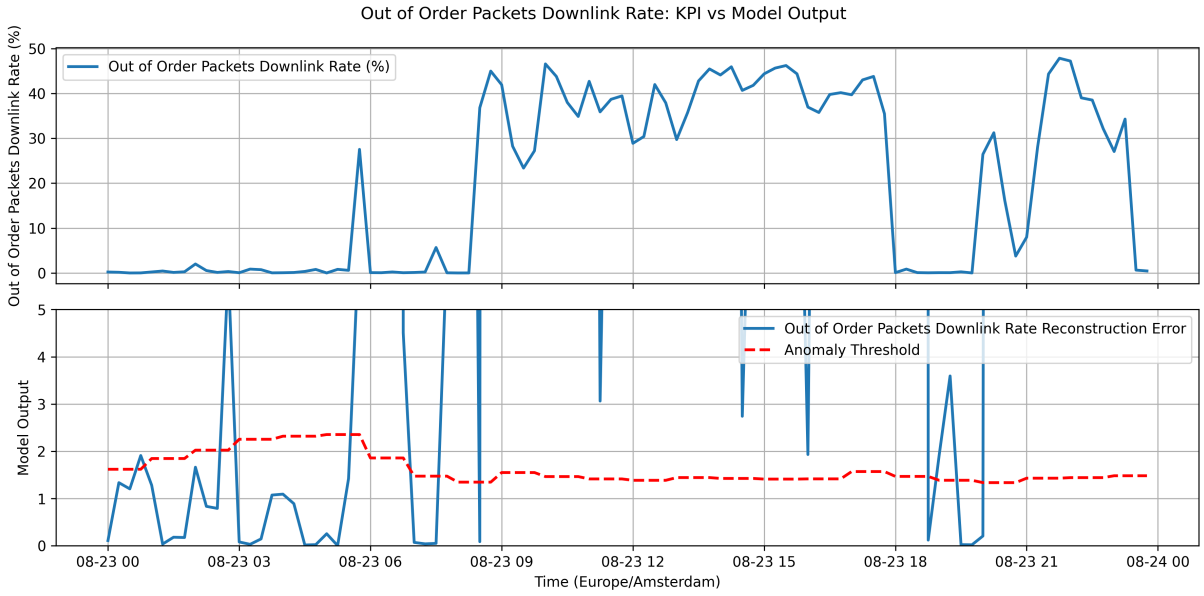

Figure 5.13: Comparison of Out of order packets downlink (top) and model output (bottom) over time zoomed in

5.1.1. Alternative validation

To quantitatively assess the behaviour of the LSTM autoencoder in the absence of labeled anomalies, the validation framework was applied across test dates 2 and 3 October 2025. The results reveal both strengths and areas requiring attention in the anomaly detection model.

Figure 5.14 presents average anomaly rates by domain, showing significant variation from 5.9% for

coreGx to 18.6% for coreRx. This high anomaly rate, particularly in coreRx and ran (same as ran_l), suggests that the anomaly detection threshold may be set too low, leading to a high number of potential false positives.

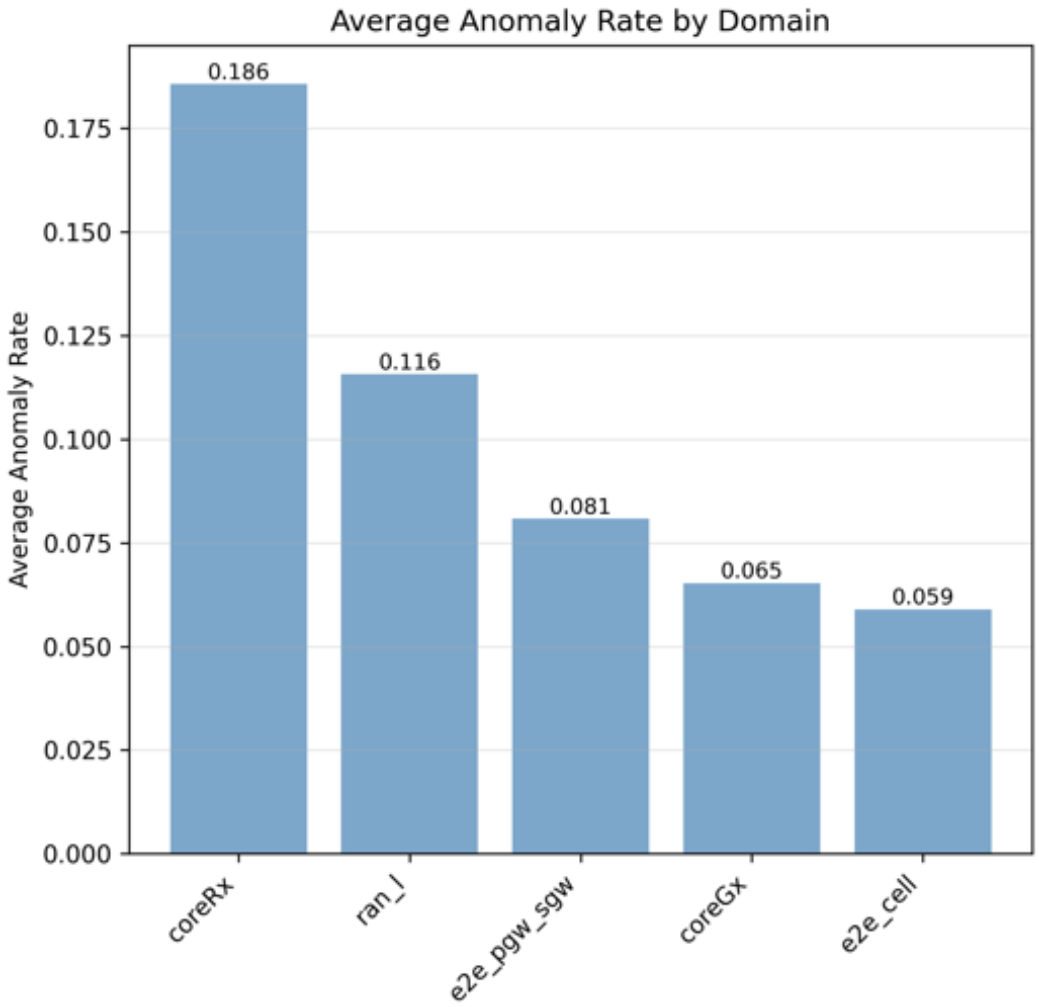

Figure 5.14: Average anomaly rate by domain

Figure 5.15 shows the pattern consistency scores, which measure the temporal coherence of detected anomalies. All domains displays high consistency scores, between 0.7 and 0.8. These scores indicate that anomalies are not random. For instance, anomalies often appear as sustained deviations during specific periods, e.g., peak traffic hours. This structured behaviour demonstrates the LSTM autoencoder's ability to capture underlying temporal dependencies and suggests that the detected anomalies correspond to systematic deviations rather than noise.

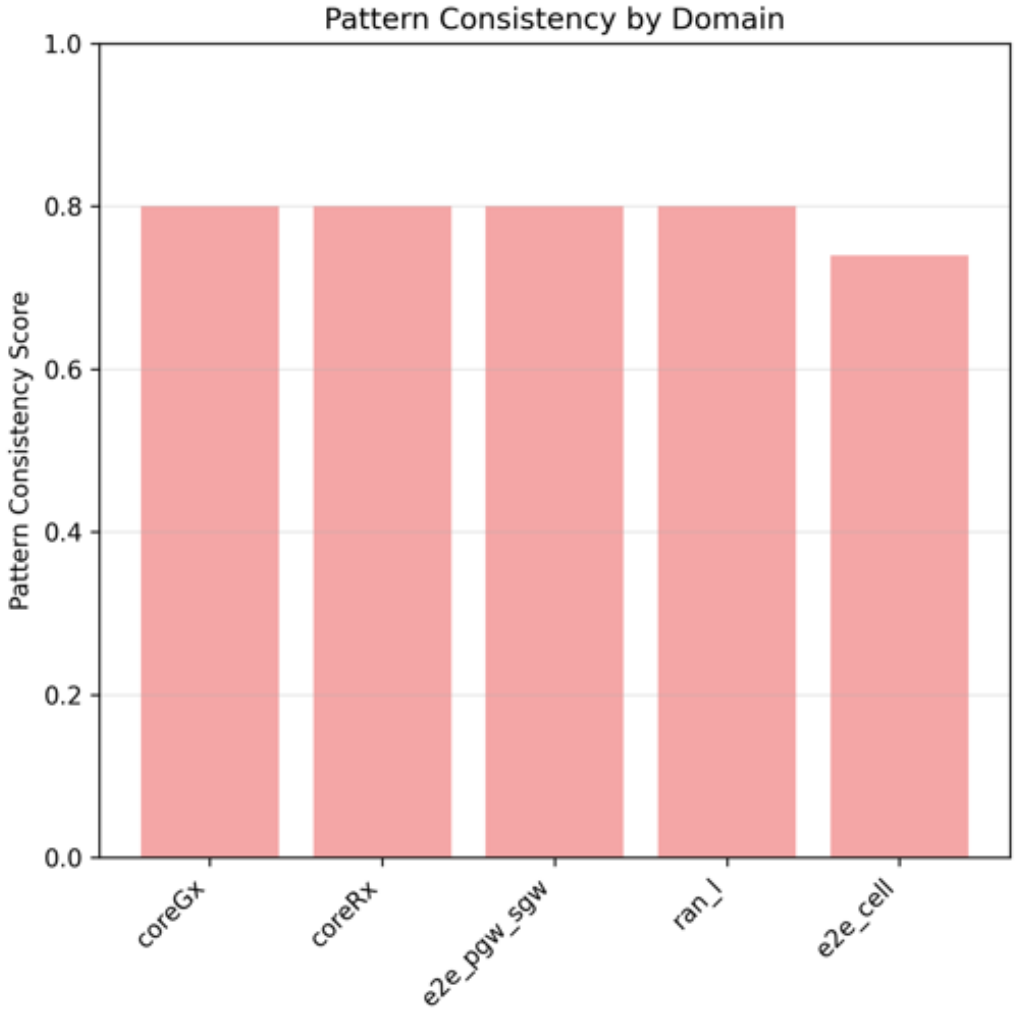

Figure 5.15: Pattern consistency by domain

Figure 5.16 shows the visual inspection that was performed using standard deviation as explained in section 4.1.2.2.8 to estimate the proportion of detected anomalies that would be noticeable to a human operator. For this only a sample of 20% of all anomalies per feature is used. Despite the high pattern consistency in Figure 5.15, only 3-4% of the anomalies are visible by human eye. This discrepancy could indicate that there are statistically significant deviations that may not be immediately obvious through visual inspection or the operator might overlook. However this could also mean that there are false positives or simply that the threshold is set too low.

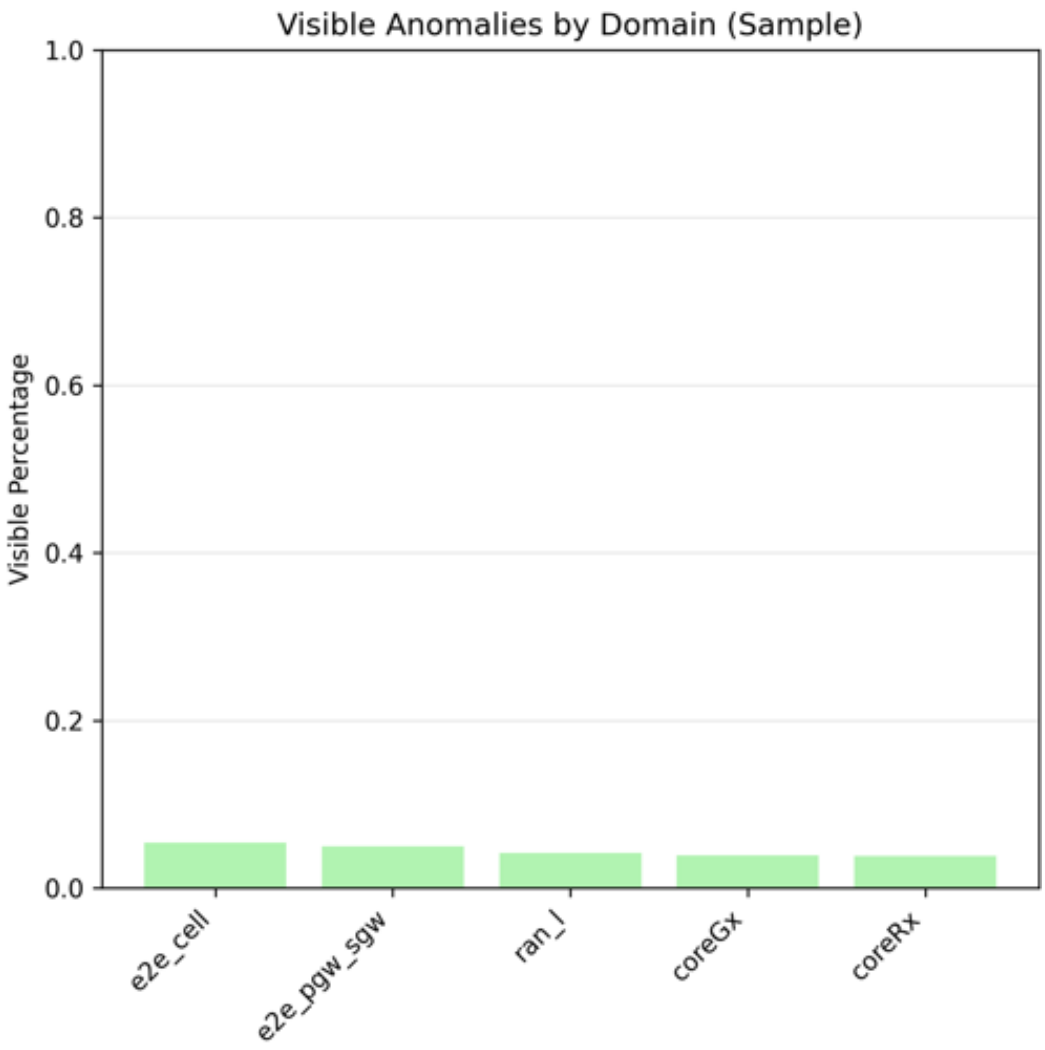

Figure 5.16: Visible anomalies by domain

The validation results demonstrate that the LSTM autoencoder successfully captures temporally consistent anomalies across all domains, with high pattern consistency scores. However, the high detection rates with mean 10.1% and very low visibility of 3 to 4% indicates that the model is overly sensitive to subtle deviations and therefore causing a high false positive rate. In order to resolve this issue, the detection threshold should be set higher to reduce the anomaly rate, while preserving the model's ability to identify meaningful temporal patterns.

5.2. Reinforcement Learning

This section presents the experimental results of the proposed offline reinforcement learning approach for root cause analysis. The model was trained on 4231 anomaly events for one day which is a very small sample of all anomalies and evaluated on an independent test set of 4297 events of 1 day. The RL agent's performance is compared against a simple heuristic baseline that selects the temporally earliest anomaly within the preceding 15 minutes as the root cause.

The Conservative Q-Learning agent was trained for 50 epochs on the offline dataset. Figure 5.17 shows the total training loss per epoch. The loss consists of both the temporal-difference (TD) regression term and the CQL penalty term. An initial sharp reduction in the training loss which is followed by an oscillatory behaviour between approximately 6.868 and 6.870. This indicates that the training has stabilized. This stabilization at a low loss value indicates that the reward scaling factor of 1/200

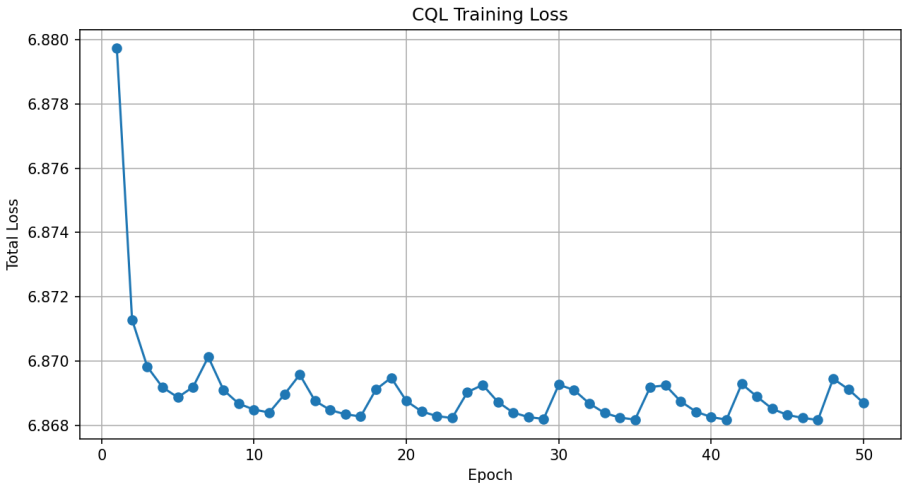

Figure 5.17: The training loss of RL

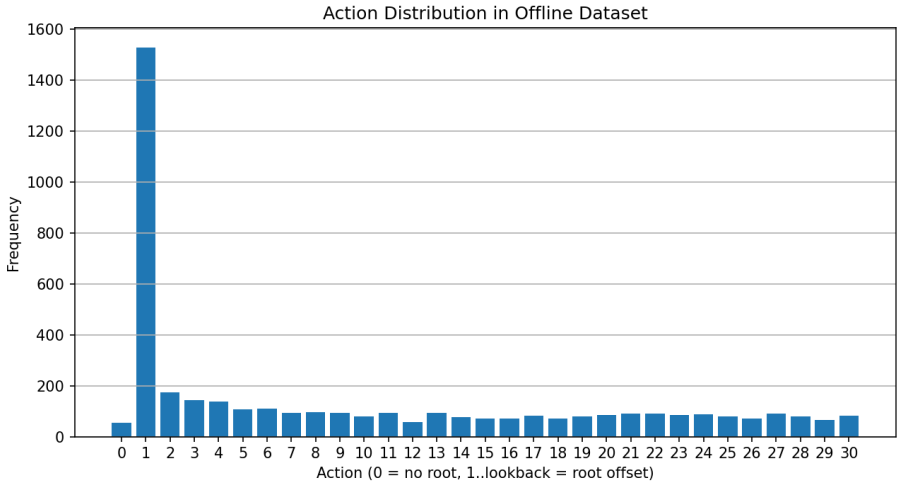

Figure 5.18: Action distribution

successfully kept gradients within a small range. Without such scaling, the raw rewards would produce much larger loss values and coudl cause gradient explosion or highly unstable training. Conversely, scaling by a too large factor would shrink gradients, causing vanishing gradients and extremely slow convergence. Scaling by too small a factor might still leave rewards too large, risking instability. The choice of 1/200 was determined to keep the average reward magnitude near 0.1-1.0, which is a normal range for neural network optimizers using standard learning rates.

Figure 5.18 shows the action distribution plot with strong bias toward action equals 1, which corresponds to selecting the immediately preceding anomaly event as the root cause. This action appears more than 1,400 times, far outnumbering all other choices. No actions, which indicates no root causes and actions 2 through 30 each occur between approximately 100 and 200 times. This imbalance is a direct consequence of the mixed behaviour policy used to generate the offline dataset: 40% of transitions are random, 30% follow a "earliest" heuristic that often picks the most recent high-error event and 30% choose the earliest event within 60 minutes, which also favours small offsets. As a result, the training data mostly shows the agent to short root cause assignments, which may bias the learned Q-function toward preferring immediate predecessors and could limit its ability to generalise to longer causal chains.

The reward distribution seen in Figure 5.19 shows a large peak near 0. The peak frequency exceeds

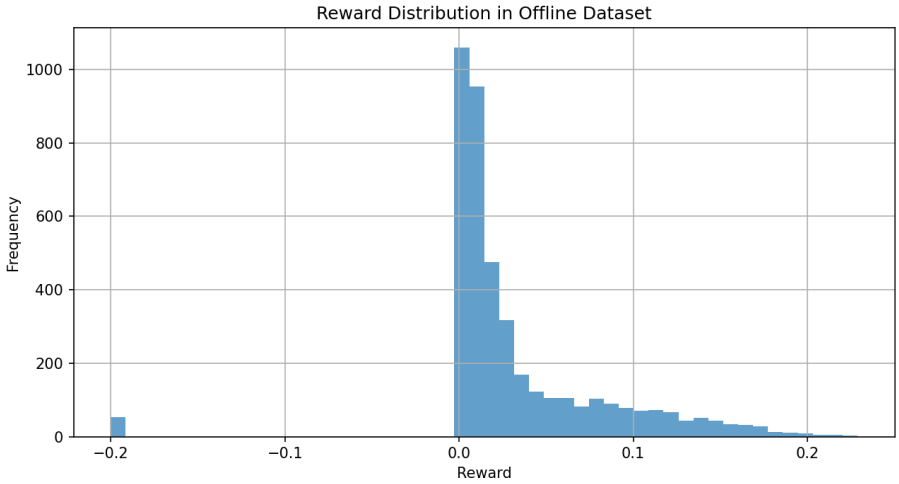

Figure 5.19: Reward distribution

1000 transitions, indicating that short root cause assignments relating to action 1, consistently yield small positive returns. This aligns with the action distribution bias, where immediate predecessors dominate the dataset. The positive skew suggests that the reward function successfully identifies causal relationships, as most transitions receive non-negative feedback. A smaller but distinct peak at -0.2 corresponds to the fixed penalty for taking action 0, which corresponds to no root cause. However, the number of high reward samples may limit the agent to strong causal signals, potentially under weight longer chains or cross-domain correlations that could yield larger cumulative rewards.

The bar chart comparing the RL agent (blue) against the earliest within-15-minute baseline (orange) reveals that both methods assign a root cause to nearly 100% of test events. The success rate of the proportion of assigned roots that pass statistical validation reaches approximately 36% for the RL agent, slightly lower than the 35% of the baseline. The average confidence of valid root causes is essentially 100% for both approaches. These results indicate that the RL agent matches the baseline in coverage and confidence while achieving a modest improvement in the accuracy of identified causal chains. The high confidence values with relatively low success rate shows the difficulty of the task and need further refinement.

Analysis of the domains from which root causes originate shows that e2e_cell is the most frequent source accounting for approximately 600 identified root causes. ran_l follows with around 550, coreRx with around 150, while e2e_pgw_sgw and coreGx each contribute about 100.

Figure 5.20: Comparison of the RL model vs Baseline

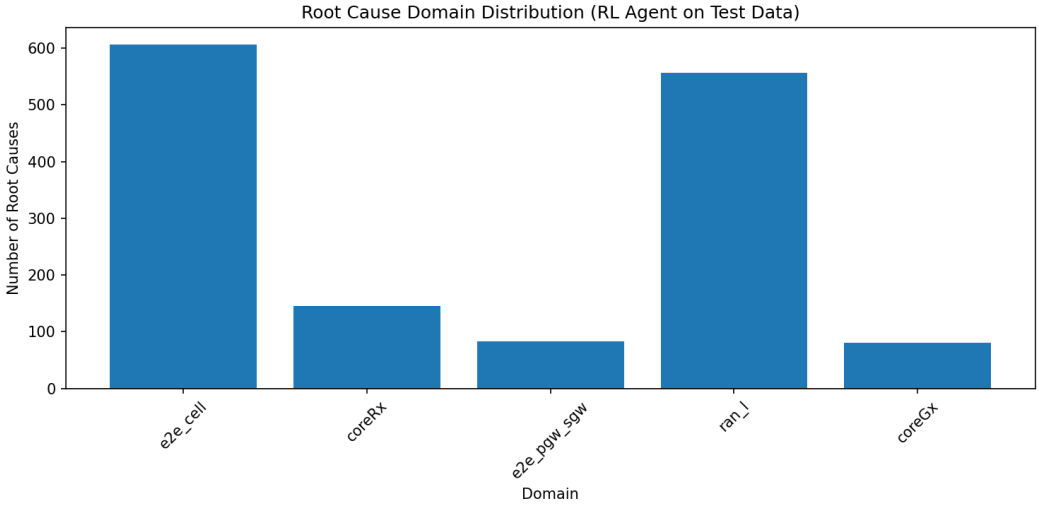

Figure 5.21: Root cause domain comparison

6
Conclusion

This thesis investigated how unsupervised anomaly detection and reinforcement learning can be combined to enable near real-time predictive fault localization in complex, multi-domain mobile networks with minimal reliance on labeled data. The primary objective was to design and evaluate a framework capable of detecting anomalous behaviour and locate probable fault location.

In this thesis, an unsupervised anomaly detection has been implemented using LSTM autoencoder. The results show that unsupervised anomaly detection is identifying deviations from normal network behaviour across multiple domains without requiring labeled fault examples. These detected anomalies provide meaningful early indicators of potential faults and can be leveraged as signals for the fault localization.

The anomaly detection and localization performance is not perfect and remain sensitive to several factors. KPI that uses ratio such as success rates are more influenced by traffic volume. During low traffic periods, such as night time, individual session failures affect success ratios, whereas similar failures during high traffic periods have minimal impact. During preprocessing, samples with no recorded attempts were filled with values of 0 or 100, depending on the interpretation of the counter. However, this approach may obscure genuine failure events, since filling default values can prevent actual faults from being accurately captured. Additionally, modern mobile networks have occasional session failures which are unavoidable due to device diversity, user behaviour or transient network conditions.

Building on these anomaly signals, an offline reinforcement learning agent was developed to perform root cause analysis by learning to associate temporally related anomalies into causal chains. The agent was trained on a relatively small dataset generated from historical anomaly sequences using a mixed behaviour policy, and evaluated against a heuristic baseline on test data. The RL agent achieved comparable performance to the baseline, with a slightly higher success rate in identifying statistically validated causal relationships, while maintaining similar assignment coverage and confidence levels.

The training revealed that the loss function remained stable. The action distribution in the training data exhibited a strong bias toward short anomalies, which may limit the agent's ability to generalize to longer causal chains. Domain analysis identified e2e_cell as the most frequent root cause source, reflecting both anomaly volume and causal influence patterns.

Future work should focus on improved data representation, adaptive thresholding strategies, enhanced handling of traffic dependent noise and refined reinforcement learning hyperparameters to move closer to operational deployment, as elaborated in Section 6.2. Despite these limitations, this thesis demonstrates that combining unsupervised anomaly detection with offline reinforcement learning offers a promising path toward automated root cause analysis in complex network environments, reducing reliance on expensive labeled data while providing actionable insights for network operations.

6.1. Discussion

The results of this thesis provide important trade-offs and limitations of predictive fault localization using unsupervised learning and reinforcement learning in operational mobile networks.

A key trade off is the number of anomaly recall and precision in the anomaly detection. When missed anomalies are costly, such as faults that may propagate rapidly across domains and impact large numbers of users, the system should prioritize high recall, accepting a lower precision and consequently generating more alerts. Conversely, when false alarms result in significant operational cost, higher detection thresholds may be preferred, resulting in fewer but more reliable alerts at the expense of recall. Therefore, there is no optimal recall or precision, but it is highly dependent on where the emphasis is

placed.

Traffic volume plays a critical role in the interpretation of KPIs with ratios. During night time periods with low traffic, a single failed session can significantly degrade success ratios, generating noise into the anomaly detection process. While excluding night time data could reduce false positives, doing so would reduce model consistency and limit its applicability across all operating conditions.

Another limitation arises from data representation. In the current implementation, missing KPI values are filled with zeros and cause ambiguity. A zero value may represent either the absence of data or a genuine failure event, such as a failed session followed by an immediate successful reestablishment. This ambiguity reduces the model's ability to distinguish between different operational states and may lead to producing incorrect anomaly. Explicitly encoding missing data or introducing additional state indicators would improve interpretability and localization accuracy.

The choice of normalization, threshold values and KPI units also significantly affects detection performance. Experimenting with alternative normalization strategies or including various weights for each KPI could further improve robustness. Additionally, although the LSTM autoencoder generates reconstruction errors as the primary anomaly score, the validation does not explicitly verify whether detected anomalies consistently correspond to significantly elevated reconstruction errors. No analysis was performed to compare reconstruction error distributions between detected anomalous and non-anomalous samples. Without this validation step, it remains unclear whether the anomaly detection thresholds are optimally aligned with the model's internal reconstruction behaviour, potentially allowing both false positives and false negatives to persist.

The reinforcement learning component introduces complex challenges. Unlike classical RL settings where an environment is a well defined reward function and ground truths are available, the fault localization problem in operational mobile networks lacks explicit labels and verified causal ground truth. The agent does not receive direct confirmation that a predicted root cause is correct. Instead, rewards must be constructed indirectly from statistical relationships and temporal consistency, making the learning signal inherently weak and noisy. Moreover, although cross-domain anomaly relationships are inferred by the agent, there is no explicit validation mechanism to confirm whether these cross-domain links represent true causal interactions or merely temporal correlations. The framework does not include statistical cross-domain validation tests or independent verification of inferred causal chains. Consequently, the learned relationships may reflect patterns in historical data without guaranteeing fault propagation. Furthermore, the available dataset is relatively small and highly imbalanced, with limited diversity in anomaly sequences and strong bias toward certain domains and short duration events. This restricts exploration of the state-action space and increases the risk of overfitting to historical patterns. Offline reinforcement learning further compounds the difficulty, as the agent cannot interact with the live environment to gather additional experience or correct suboptimal policies. As a result, convergence becomes unstable, training loss oscillations are more likely, and performance improvements over heuristic baselines are naturally constrained. These limitations highlight that applying reinforcement learning in this context is exceptionally difficult and requires careful reward design and regularization strategies to ensure meaningful and stable learning.

Another practical limitation relates to the anomaly segmentation behaviour of the autoencoder. In certain cases, the model detects only the beginning and ending points of an anomalous period rather than continuously marking the entire duration of abnormal behaviour. This fragmented detection complicates root cause analysis, as reinforcement learning relies on temporally anomaly sequences. Short anomaly bursts may dominate the training data, reducing the agent's ability to learn extended causal chains and making persistent degradations harder to interpret operationally.

6.2. Future works

While the proposed framework demonstrates the feasibility of combining unsupervised anomaly detection and offline reinforcement learning for root cause analysis in mobile networks, there are still multiple opportunities for improvement. Future research should address the following directions to enhance robustness and operational readiness.

- Alternative normalization techniques to evaluate whether different scaling strategies improve

model stability and performance.

- Different temporal window sizes, such as 90-minute intervals instead of one hour. Shorter or longer durations could be explored to determine the optimal temporal resolution for anomaly detection in mobile networks.
- Deploying the framework on servers or GPUs could enable the use of deeper model architectures and accelerate anomaly detection processing, thereby improving computational efficiency and scalability.
- The framework could be integrated into existing network management systems to support near-real-time operational use.
- Geographical information such as antenna directionality and relationships between cells could be incorporated into the models to enhance contextual understanding and root cause localization.
- Separate models could be trained for specific time periods, e.g., working hours versus night time. Night time periods typically involve lower traffic volumes, which can introduce higher variance and noise in ratio-based KPIs. Training dedicated models for these periods may therefore improve stability and anomaly detection accuracy.
- Certain network periods, such as major planned events, e.g., large public events or network freeze periods, may be more stable for training due to no network configuration changes are allowed. So saving data on those days and using it for training might improve the training.
- Additional KPIs could be incorporated into the framework to enrich the feature space and provide a more comprehensive representation of network behaviour.
- If labeled data becomes available, it could be leveraged to better train and calibrate the offline reinforcement learning component using ground-truth root cause information, thereby improving policy learning and decision accuracy.
- Weekly and monthly temporal patterns could be incorporated in the model to capture long-term trends and recurring behaviours that may not be visible within shorter time windows.

References

[1] IJIRSET. "6G Network and Challenges". In: *International Journal of Innovative Research in Science, Engineering and Technology (IJIRSET)* 13.5 (May 2024). DOI: 10.15680/IJIRSET.2024.1305582. URL: https://www.ijirset.com/upload/2024/may/582_6G.pdf.

[2] Wanshi Chen et al. "5G-Advanced Toward 6G: Past, Present, and Future". In: *IEEE Journal on Selected Areas in Communications* 41.6 (June 2023), pp. 1592–1619. DOI: 10.1109/JSAC.2023.3274037.

[3] *Ericsson Mobility Report*. Ericsson, Nov. 2023. URL: https://www.ericsson.com/en/reports-and-papers/mobility-report/reports.

[4] *The Mobile Economy 2024*. GSMA, Feb. 2024. URL: https://www.gsma.com/mobileeconomy/wp-content/uploads/2024/02/260224-The-Mobile-Economy-2024.pdf.

[5] 3rd Generation Partnership Project (3GPP). *5G; Management and orchestration; Architecture framework (3GPP TS 28.533 version 18.1.0 Release 18)*. Technical Specification ETSI TS 128 533 V18.1.0. ETSI, May 2024. URL: https://www.etsi.org/deliver/etsi_ts/128500_128599/128533/18.01.00_60/ts_128533v180100p.pdf.

[6] International Telecommunication Union. *Information technology – Open Systems Interconnection – Systems Management: Alarm Reporting Function*. CCITT Recommendation X.733 | ISO/IEC 10164-4. Also published as ISO/IEC 10164-4:1992. Geneva: International Telecommunication Union, 1992.

[7] M. Manzano et al. "Unveiling Potential Failure Propagation Scenarios in Core Transport Networks". In: *arXiv preprint arXiv:1402.2680* (2014). arXiv: 1402.2680 [cs.NI]. URL: https://arxiv.org/abs/1402.2680.

[8] Alexander Clemm. *Network Management Fundamentals*. Indianapolis, IN, USA: Cisco Press, 2007. ISBN: 1-58720-137-2. URL: https://cs.petrsu.ru/~vadim/books/Network%20Management%20Fundamentals.pdf.

[9] Małgorzata Steinder and Adarshpal S. Sethi. "A survey of fault localization techniques in computer networks". In: *Science of Computer Programming* 53.2 (Nov. 2004), pp. 165–194. DOI: 10.1016/j.scico.2004.01.010. URL: https://doi.org/10.1016/j.scico.2004.01.010.

[10] J. L. Silitonga. "A Review of AI-Driven Predictive Maintenance in Telecommunications". In: 3.2 (Dec. 2024), pp. 25–31. DOI: 10.63322/tsq25y55. URL: https://doi.org/10.63322/tsq25y55.

[11] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41 (July 2009). DOI: 10.1145/1541880.1541882.

[12] Yaswanth Kumar L S et al. *FALCON: A Framework for Fault Prediction in Open RAN Using Multi-Level Telemetry*. arXiv preprint arXiv:2503.06197. IEEE INFOCOM Workshop on Shaping the Future of Telecoms: Networks for Joint Intelligence, Sustainability, Security, and Resilience 2025. 2025. DOI: 10.48550/arXiv.2503.06197. arXiv: 2503.06197 [cs.NI]. URL: https://arxiv.org/abs/2503.06197.

[13] 3GPP SA5. *3GPP TS 28.105: Management and orchestration; Management of 5G networks; Machine Learning function*. Tech. rep. Version 17.4.0. 3rd Generation Partnership Project, 2023. URL: https://www.etsi.org/deliver/etsi_ts/128100_128199/128105/17.04.00_60/ts_128105v170400p.pdf.

[14] ETSI ISG ENI. *ETSI GR ENI 005: Experiential Networked Intelligence (ENI); System Architecture*. Tech. rep. ETSI, 2021. URL: https://www.etsi.org/deliver/etsi_gs/ENI/001_099/005/02.01.01_60/gs_ENI005v020101p.pdf.

[15] C. Chang. "Overcoming Data Scarcity in Deep Learning of Scientific Problems". PhD thesis. Massachusetts Institute of Technology, 2021. URL: https://hdl.handle.net/1721.1/140165.

[16] L. Zhao, C. He, and X. Zhu. "A Fault Diagnosis Method for 5G Cellular Networks Based on Knowledge and Data Fusion". In: *Sensors* 24.2 (Jan. 2024), pp. 401–401. DOI: 10.3390/s24020401. URL: https://doi.org/10.3390/s24020401.

[17] Amin Yousefpour et al. "Unsupervised Anomaly Detection via Nonlinear Manifold Learning". In: *arXiv preprint arXiv:2306.09441* (2023). arXiv: 2306.09441 [cs.LG]. URL: https://arxiv.org/html/2306.09441.

[18] Ping Liu et al. "FluxRank: A Widely-Deployable Framework to Automatically Localizing Root Cause Machines for Software Service Failure Mitigation". In: *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*. Oct. 2019, pp. 35–46. DOI: 10.1109/ISSRE.2019.00014. URL: https://ieeexplore.ieee.org/document/8987478.

[19] I. Ahmed, M. Quiñones-Grueiro, and G. Biswas. "Fault-Tolerant Control of Degrading Systems with On-Policy Reinforcement Learning". In: *IFAC-PapersOnLine* 53.2 (Apr. 2021), pp. 13733–13738. DOI: 10.1016/j.ifacol.2020.12.878.

[20] Enzo Guo, James Lin, and Yang Zhang. *THE PERFORMANCE ANALYSIS OF LTE NETWORK*. Tech. rep. Simon Fraser University, 2014. URL: https://www.sfu.ca/~ljilja/ENSC427/Spring14/Projects/team6/ENSC427_team6_report.pdf.

[21] Ayman Elnashar. *Practical Aspects of LTE Network Design and Deployment*. Tech. rep. arXiv, 2018. URL: https://arxiv.org/pdf/1810.02970.pdf.

[22] MPC Courses. *Network Architecture*. Lesson on LTE Evolved Packet Core, including control/user plane, MME, SGW, PGW, and HSS functions. 2025. URL: https://mpc-courses.teachable.com/courses/learn-4g-lte-mobile-packet-core-in-3-hrs/lectures/45721260.

[23] Erik Dahlman, Stefan Parkvall, and Johan Sköld. *4G: LTE/LTE-Advanced for Mobile Broadband*. Elsevier, 2014. DOI: https://doi.org/10.1016/C2013-0-06829-6.

[24] Devopedia. *Control and User Plane Separation*. Version 4. Accessed: 2024-06-25. Aug. 2023. URL: https://devopedia.org/control-and-user-plane-separation.

[25] Petar Popovski et al. "5G Wireless Network Slicing for eMBB, URLLC, and mMTC: A Communication-Theoretic View". In: *arXiv preprint arXiv:1804.05057* (2018). URL: https://arxiv.org/abs/1804.05057.

[26] Ericsson. *Massive MIMO solutions accelerate 5G mid-band*. 2024. URL: https://www.ericsson.com/en/ran/massive-mimo.

[27] J. Ordonez-Lucena et al. "Network Slicing for 5G with SDN/NFV: Concepts, Architectures, and Challenges". In: *IEEE Communications Magazine* 55.5 (May 2017), pp. 80–87. DOI: 10.1109/MCOM.2017.1600935. URL: https://doi.org/10.1109/mcom.2017.1600935.

[28] Panagiotis K. Syriopoulos et al. "kNN Classification: a review". In: *Annals of Mathematics and Artificial Intelligence* (Sept. 2023). DOI: 10.1007/s10472-023-09882-x. URL: https://doi.org/10.1007/s10472-023-09882-x.

[29] Markus M. Breunig et al. "LOF: Identifying Density-Based Local Outliers". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. ACM, 2000, pp. 93–104. URL: https://www.researchgate.net/publication/221214719_LOF_Identifying_Density-Based_Local_Outliers.

[30] F. T. Liu, K. M. Ting, and Z.-H. Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422. DOI: 10.1109/icdm.2008.17. URL: https://ieeexplore.ieee.org/document/4781136.

[31] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv preprint arXiv:1312.6114* (2013). DOI: https://doi.org/10.48550/arXiv.1312.6114. URL: https://arxiv.org/pdf/1312.6114.

[32] Thomas Schlegl et al. "f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks". In: *Medical Image Analysis* 54 (May 2019), pp. 30–44. DOI: 10.1016/j.media.2019.01.010. URL: https://www.sciencedirect.com/science/article/abs/pii/S1361841518302640?via%3Dihub.

[33] Ira Golan and Sharon Elbaum. "Deep Anomaly Detection Using Geometric Transformations". In: *Advances in Neural Information Processing Systems* 31 (2018). NeurIPS 2018. URL: https://arxiv.org/abs/1805.10917.

[34] Xiaoxiao Ma et al. "A Comprehensive Survey on Graph Anomaly Detection with Deep Learning". In: *arXiv preprint arXiv:2106.07178* (June 2021). DOI: 10.48550/arXiv.2106.07178. URL: https://doi.org/10.48550/arXiv.2106.07178.

[35] Jiaxi Li et al. "HRGCN: Heterogeneous Graph-level Anomaly Detection with Hierarchical Relation-augmented Graph Neural Networks". In: *arXiv preprint arXiv:2308.14340* (2023). arXiv: 2308.14340 [cs.LG]. URL: https://arxiv.org/abs/2308.14340.

[36] Miro-Markus Nikula et al. "Machine Learning-Based Anomaly Detection and Root Cause Analysis in Mobile Networks". In: *Research Portal of the University of Lisbon* (2025). URL: https://researchportal.ulisboa.pt/en/publications/machine-learning-based-anomaly-detection-and-root-cause-analysis-.

[37] Martijn van Otterlo and Marco Wiering. "Reinforcement Learning and Markov Decision Processes". In: *Reinforcement Learning*. Ed. by Marco Wiering and Martijn van Otterlo. Springer, 2012, pp. 3–42. URL: https://www.ai.rug.nl/~mwiering/Intro_RLBOOK.pdf.

[38] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518 (2015), pp. 529–533. DOI: 10.1038/nature14236. URL: https://doi.org/10.1038/nature14236.

[39] John Schulman et al. "Proximal Policy Optimization Algorithms". In: *arXiv preprint arXiv:1707.06347* (Aug. 2017). DOI: 10.48550/arXiv.1707.06347. URL: https://arxiv.org/abs/1707.06347.

[40] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd. Cambridge, MA: MIT Press, 2018. ISBN: 9780262039246. URL: http://incompleteideas.net/book/the-book-2nd.html.

[41] Chen Ran et al. "Policy Regularization with Dataset Constraint for Offline Reinforcement Learning". In: *Proceedings of the Twelfth International Conference on Learning Representations* (2023). arXiv:2306.06569 [cs.LG]. arXiv: 2306.06569 [cs.LG]. URL: https://arxiv.org/abs/2306.06569.

[42] Alan Oursland. "Neural Networks Learn Distance Metrics". In: (2025). arXiv: 2502.02103 [cs.LG]. URL: https://arxiv.org/html/2502.02103v1.

[43] Juyoung Yun et al. "ZNorm: Z-Score Gradient Normalization for Accelerating Neural Network Training". In: (2024). arXiv: 2408.01215 [cs.LG]. URL: https://arxiv.org/html/2408.01215v1.

[44] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[45] Oluwaseun S. Oyerinde and Bamidele Adebisi. "Performance Analysis of KPI's of a 4G Network in a Specific Location". In: *International Journal of Communications* 15.2021 (2021), pp. 005–0004. URL: https://www.iaras.org/iaras/filedownloads/ijoc/2021/005-0004(2021).pdf.

[46] *pandas: Python Data Analysis Library*. https://pandas.pydata.org/.

[47] *Polars: DataFrames for the New Era*. https://pola.rs/.

[48] A. Getis. "Spatial Pattern Analysis". In: *Elsevier eBooks*. Elsevier, Jan. 2005, pp. 627–632. DOI: 10.1016/b0-12-369398-5/00336-4.

[49] Hasim Sak, Andrew Senior, and Françoise Beaufays. *Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling*. Tech. rep. Google, 2014. URL: https://research.google.com/pubs/archive/43905.pdf.

[50] scikit-learn developers. *Time-related feature engineering*. https://scikit-learn.org/stable/auto_examples/applications/plot_cyclical_feature_engineering.html. Accessed: 2025-12-23. 2024.

[51] Jiawei Yang, Susanto Rahardja, and Pasi Fränti. "Outlier Detection: How to Threshold Outlier Scores?" In: *Proceedings of the 2019 International Conference on Artificial Intelligence and Pattern Recognition (AIIPCC 2019)*. New York, NY, USA: Association for Computing Machinery, 2019, pp. 1–10. ISBN: 978-1-4503-7633-4. DOI: 10.1145/3371425.3371427. URL: https://doi.org/10.1145/3371425.3371427.

[52] Muyan Anna Li and Aditi Gautam. “Segmented Confidence Sequences and Multi-Scale Adaptive Confidence Segments for Anomaly Detection in Nonstationary Time Series”. In: *arXiv preprint arXiv:2508.06638* (2025). DOI: 10.48550/arXiv.2508.06638. arXiv: 2508.06638 [cs.LG]. URL: https://arxiv.org/abs/2508.06638.

[53] Guillaume A. Rousselet and Cyril R. Pernet. “Improving standards in brain-behavior correlation analyses”. In: *Frontiers in Human Neuroscience* 6 (2012), p. 119. ISSN: 1662-5161. DOI: 10.3389/fnhum.2012.00119. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC3342588/.

[54] MM Mukaka. “A guide to appropriate use of Correlation coefficient in medical research”. In: *Malawi Medical Journal* 24.3 (2012), pp. 69–71. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC3576830/.