



UNIVERSIDADE FEDERAL DO RIO GRANDE - FURG
CENTRO DE CIÊNCIAS COMPUTACIONAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM MODELAGEM COMPUTACIONAL
CURSO DE MESTRADO EM MODELAGEM COMPUTACIONAL

Dissertação de Mestrado

Combining Entropy Metrics and Autoencoder-LSTM Models for Lightweight Data Injection Detection in ROS 2

Jorge Gutiérrez

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Modelagem Computacional da Universidade Federal do Rio Grande - FURG, como requisito parcial para a obtenção do grau de Mestre em Modelagem Computacional

Advisor: Prof. Dr. Paulo Lilles Jorge Drews Junior
Co-advisor: Prof. Dr. Bruno Lopes Dalmazo

Rio Grande, 2026

		0
G984c	Gutiérrez, Jorge Combining Entropy Metrics and Autoencoder-LSTM Models for Lightweight Data Injection Detection in ROS 2 /Jorge Gutiérrez – 2026. 91f. Dissertação (Mestrado) – Universidade Federal do Rio Grande – Programa de Pós-Graduação em Modelagem Computacional, 2026. Orientador: Dr Paulo Lilles Jorge Drews Junior Co-orientadora: Dr. Bruno Lopes Dalmazo 1. Modelagem Computacional. 2. ROS 2. 3. Detecção de anomalias. 4. Entropia 5. Autoencoder. 6.LSTM. I. Drews Junior, Paulo Lilles Jorge. II. Dalmazo, Bruno Lopes. III. Título.	
		CDU 004.414.23

Catálogo na fonte: Bibliotecária Brenda Sequeira CRB10/1979

ATA DE DEFESA DE DISSERTAÇÃO**ATA Nº 07/2025**

Aos 31 dias do mês de outubro de dois mil e vinte e cinco, por meio de videoconferência, realizou-se a 191ª sessão de defesa de dissertação do Programa de Pós-Graduação em Modelagem Computacional. O mestrando **Jorge Nelson Gutiérrez Pena** apresentou e defendeu a dissertação **“Combining Entropy Metrics and Autoencoder-LSTM Models for Lightweight Data Injection Detection in ROS 2”** como requisito parcial para a obtenção do grau de Mestre em Modelagem Computacional. A sessão foi aberta às quatorze horas pelo Prof. Dr. Paulo Lilles Jorge Drews Junior, orientador e presidente da Comissão de Avaliação. A referida Comissão esteve integrada, além do presidente e orientador, pelo Prof. Dr. Bruno Lopes Dalmazo, co-orientador (FURG), pelo Prof. Dr. João Alberto Fabro (UTFPR), Prof. Dr. Emanuel da Silva Diaz Estrada (FURG) e pela Profa. Dra. Thaísa Leal da Silva (ATITUS). Após a apresentação do trabalho, a banca arguiu o candidato e, a seguir, deliberou pela

- (X) aprovação da Dissertação
() aprovação da Dissertação, sugerindo modificações no texto
() reprovação da Dissertação

Assinado de forma digital por
PAULO LILLES JORGE DREWS
JUNIOR:00124140009
Dados: 2025.12.22 11:51:22
-03'00'**Prof. Dr. Paulo Lilles Jorge Drews Junior**
OrientadorDocumento assinado digitalmente
THAISA LEAL DA SILVA
Data: 24/12/2025 12:29:24-0300
Verifique em <https://validar.iti.gov.br>**Profa. Dra. Thaísa Leal da Silva**
Membro ExternoDocumento assinado digitalmente
BRUNO LOPES DALMAZO
Data: 24/12/2025 12:34:15-0300
Verifique em <https://validar.iti.gov.br>**Prof. Dr. Bruno Lopes Dalmazo**
CoorientadorDocumento assinado digitalmente
EMANUEL DA SILVA DIAZ ESTRADA
Data: 24/12/2025 12:52:59-0300
Verifique em <https://validar.iti.gov.br>**Prof. Dr. Emanuel da Silva Diaz Estrada**
Membro InternoDocumento assinado digitalmente
JOAO ALBERTO FABRO
Data: 22/12/2025 17:55:42-0300
Verifique em <https://validar.iti.gov.br>**Prof. Dr. João Alberto Fabro**
Membro Externo

"The world is not threatened by evil people, but by those who allow evil"—

ALBERT EINSTEIN

AGRADECIMENTOS

Expresso minha profunda gratidão aos meus orientadores, Prof. Dr. Paulo Lilles Jorge Drews-Jr. e Prof. Dr. Bruno Lopes Dalmazo, pela orientação rigorosa e generosa ao longo de todo este processo. Agradeço também ao corpo docente do programa por sua contribuição para a minha formação e aos membros da banca examinadora, Prof. Dr. Emanuel Estrada, Prof. Dr. João Alberto Fabro e Prof. Dr. Thaísa Leal da Silva, pelos comentários valiosos, que fortaleceram este trabalho e ajudaram a delinear suas direções futuras. Estendo meu reconhecimento ao Conselheiro da UTEC, Dr. Marcelo Ubal; ao Coordenador do curso de Engenharia Mecatrônica, Eng. Yamilé Lara; e à Prof. Dr Melody Garcia, Coordenadora da equipe iCiencia, pelo apoio constante e incentivo. Por fim, sou profundamente grato ao Programa de Pós-Graduação em Modelagem Computacional (PPGMC) da Universidade por tornar esta experiência possível e, de maneira muito especial, à minha família, cujo apoio incondicional tornou esta jornada possível.

RESUMO

GUTIÉRREZ, Jorge. **Combining Entropy Metrics and Autoencoder–LSTM Models for Lightweight Data Injection Detection in ROS 2**. 2026. 91 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Modelagem Computacional. Universidade Federal do Rio Grande - FURG, Rio Grande.

Garantir a segurança de sistemas robóticos distribuídos baseados em ROS 2 requer mecanismos leves capazes de detectar precocemente comportamentos anômalos sob requisitos de tempo real. Este trabalho propõe um esquema híbrido para detecção de ataques de injeção de dados que combina métricas de entropia calculadas em janelas deslizantes (entropia de Shannon, entropia de transição e entropia baseada em KDE) com um pipeline AE–LSTM treinado exclusivamente com dados nominais do robô. Primeiramente, caracteriza a estabilidade temporal das métricas e verifica a estacionariedade por meio do teste Augmented Dickey–Fuller (ADF), o que permite a calibração de limiares em segmentos nominais. Em seguida, o AE gera um espaço latente sobrecompleto que o LSTM utiliza para prever um passo à frente; a pontuação de anomalia é definida como uma combinação padronizada de forma robusta (mediana/MAD) entre o erro de reconstrução (AE) e o erro de previsão (LSTM), com uma regra de persistência K para controlar alarmes falsos.

Na avaliação com dados offline, com busca em grade de parâmetros e restrição explícita da taxa de falsos positivos (*False Positive Rate*, FPR), isto é, a proporção de amostras nominais indevidamente sinalizadas como anômalas, o ponto de operação selecionado ($\alpha = 1,1$; $\beta = 1,4$; $K = 3$) atinge Recall $\approx 0,999$ com FPR $\approx 0,097$ e $F_1 \approx 0,692$ (prevalência $\pi \approx 0,098$; $N = 7559$). Em nível de evento, obtém taxa de acerto de 100%, latência média de 0,27 janelas e cobertura média de 0,908, evidenciando detecção oportuna com taxas de falso alarme controladas. A abordagem baseia-se apenas na telemetria embarcada, não depende de inspeção de tráfego de rede e é adequada para implantação online em ROS 2, atendendo aos requisitos de tempo real.

Keywords: ROS 2, detecção de anomalias, entropia, Autoencoder, LSTM.

ABSTRACT

GUTIÉRREZ, Jorge. **Combining Entropy Metrics and Autoencoder-LSTM Models for Lightweight Data Injection Detection in ROS 2.** . 2026. 91 f. Dissertação (Mestrado) – Programa de Pós-Graduação em Modelagem Computacional. Universidade Federal do Rio Grande - FURG, Rio Grande.

Securing distributed robotic systems built on ROS 2 requires lightweight mechanisms for early, reliable detection of anomalous behavior under real-time requirements. This work proposes a hybrid scheme for detecting data-injection attacks that combines entropy metrics computed over sliding windows (Shannon, transition entropy, and KDE-based entropy) with an AE-LSTM pipeline trained exclusively on the robot's nominal data. We first characterize the temporal stability of the metrics and verify stationarity via the Augmented Dickey-Fuller (ADF) test, enabling threshold calibration on nominal segments. The AE yields an overcomplete latent space that the LSTM uses to forecast one step ahead; the anomaly score is defined as a median/MAD-standardized combination of reconstruction error (AE) and prediction error (LSTM), with a persistence rule K to control false alarms.

In the offline evaluation, we performed a grid search over parameters under an explicit constraint on the false positive rate (FPR). The selected operating point ($\alpha = 1.1$; $\beta = 1.4$; $K = 3$) achieves Recall ≈ 0.999 with FPR ≈ 0.097 and $F_1 \approx 0.692$ (prevalence $\pi \approx 0.098$; $N = 7,559$). At the event level, we obtain a 100% hit rate, a mean latency of 0.27 windows (median 0), and a mean coverage of 0.908, indicating timely detection with controlled false-alarm rates. The approach relies solely on onboard telemetry, remains agnostic to network traffic (i.e., it does not require network-traffic inspection), and is suitable for online deployment in ROS 2 while meeting real-time requirements.

Keywords: ROS2, anomaly detection, entropy, Autoencoder, LSTM.

LIST OF FIGURES

1	ROS 2 over DDS, vertical view: the client layer (rcl/rmw) maps ROS types to DDS/IDL; QoS and discovery/liveliness mechanics shape timing (inter-arrival intervals Δt , jitter). DDS Security adds authentication, access control, and encryption (NONE/SIGN/ENCRYPT) with latency/throughput impact. These signals feed the proposed detector.	20
2	Data-collector node workflow: from ROS 2 subscriptions to entropy computation, feature extraction, CSV logging, and informational logs.	35
3	Absolute change between successive samples for each entropy (Shannon, transition, and KDE).	43
4	Temporal evolution of entropy metrics (Shannon, transition, and KDE) on nominal data: raw signal, moving average, and $\pm 1\sigma$ bands. Useful to verify stability after burn-in and to guide the calibration of reference statistics.	44
5	Correlation matrix (Spearman, ρ). Color coding in $[-1, 1]$ highlights association blocks, inverse relations, and redundancies across variables.	45
6	Robustness check with Kendall (τ_b). Qualitative preservation of the main blocks detected with Spearman is observed.	46
7	Estimator-dependent differences between Spearman (ρ) and Kendall (τ_b). Left: signed map ($\rho - \tau_b$). Right: magnitude map $ \rho - \tau_b $. Differences are small overall and concentrated in a few pairs, hence the qualitative dependence structure is preserved across estimators. .	47
8	Overcomplete autoencoder used in the pipeline. The encoder maps $x_t \in \mathbb{R}^8$ to a latent embedding $z_t \in \mathbb{R}^{10}$ (MLP $8 \rightarrow 128 \rightarrow 64 \rightarrow 10$) and a symmetric decoder reconstructs $\hat{x}_t$ ($10 \rightarrow 64 \rightarrow 128 \rightarrow 8$). Dropout ($p = 0.10$) is applied in hidden layers during training, and the model is optimized with MSE reconstruction loss.	49
9	Autoencoder loss evolution during training (MSE per epoch).	50
10	Manhattan distance (MAE) per epoch during autoencoder training. . .	50
11	Residual LSTM predictor used to forecast the next latent embedding. A window of $T = 20$ <i>standardized</i> embeddings ($\mathbf{Z}_t \in \mathbb{R}^{20 \times 10}$, normalized with a <code>StandardScaler</code> fitted on the training set) is processed by a one-layer LSTM ($H = 32$). The last hidden state is mapped by a fully connected layer ($32 \rightarrow 10$) to estimate a residual increment Δ_t , and the prediction is formed as $\hat{\mathbf{z}}_{t+1} = \mathbf{z}_t + \Delta_t$, where $\mathbf{z}_t$ is the last embedding in the input window.	52

12	Loss evolution (MSE) per epoch during LSTM predictor training. . .	53
13	System-level data flow of the Anomaly Detection System (ADS) on ROS 2/DDS. Telemetry published by the ROS 2 robot is transported over the DDS network and subscribed by the ADS, which computes a detection score by combining entropy-based descriptors with an Autoencoder (AE) and an LSTM predictor. A binary anomaly decision is obtained by comparing the score with a threshold t_h under a persistence rule K . When an anomaly is detected, the ADS publishes an alert (/ads/alert) to the operator's GUI and logs. The main attack surface, where an adversary can inject or spoof DDS traffic, is indicated on the left.	55
14	End-to-end AE-LSTM flow.	56
15	High-level overview of the two-stage offline calibration procedure: Stage 1 logs raw AE/LSTM errors on the evaluation set; Stage 2 applies robust standardization, combines them into the score $S_t(\alpha, \beta)$, and sweeps thresholds and persistence K under an FPR constraint. . .	59
16	F_1 maps over (alpha,beta) for different K	63
17	Time series of the anomaly score S_t with threshold τ and K -persistence. Top: raw crossings $S_t \geq \tau$ (points) and GT episodes (shaded). Bottom: consolidated alarms after requiring K consecutive exceedances.	64
18	Two-column vertical diagram: the attacker alters data or rate; telemetry affects entropy values; the Autoencoder-LSTM scheme raises the robust score with persistence K and issues the ADS alert.	66

LIST OF TABLES

1	Comparison of entropy metrics used.	38
2	Variables captured by the data collector node.	39
3	Relationship between collected metrics and possible anomalies detected.	39
4	Stability measures for the entropy metrics.	42
5	ADF test results for entropies computed from robot behavior.	43
6	Selected operating point in <i>offline</i> calibration.	62
7	Best combination by persistence K under $\text{FPR} \leq 0.10$ (<i>offline</i>).	62
8	Event-level results (<i>offline</i> evaluation).	64
9	Executive summary of the <i>offline</i> evaluation.	67
10	Artifacts generated by the AE+LSTM pipeline and their purpose.	91

LIST OF ABBREVIATIONS AND ACRONYMS

ADS	Anomaly Detection System
AE	Autoencoder
ADF	Augmented Dickey–Fuller
AIC	Akaike Information Criterion
CSV	Comma-Separated Values
DDS	Data Distribution Service
DoS	Denial of Service
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
KDE	Kernel Density Estimation
LiDAR	Light Detection And Ranging
LSTM	Long Short-Term Memory
MAD	Median Absolute Deviation
MAE	Mean Absolute Error
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
OCSVM	One-Class Support Vector Machine
ReLU	Rectified Linear Unit
ROS 2	Robot Operating System 2
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
VAE	Variational Autoencoder

CONTENTS

1	Introduction	13
1.1	Motivation and Justification	14
1.2	Problem Description	14
1.3	Objectives	16
1.3.1	General Objective	16
1.3.2	Specific Objectives	16
1.4	Contributions	16
1.5	Structure of the Work	17
2	Theoretical Foundations	19
2.1	ROS 2: Fundamentals and QoS/Security Considerations	19
2.2	Anomaly Definition and a Data-Injection-Centered Threat Model	21
2.3	Sliding-Window Handling	22
2.4	Entropy Metrics	23
2.4.1	Shannon Entropy	23
2.4.2	Transition Entropy	24
2.4.3	KDE-Based Entropy	25
2.5	Autoencoder + LSTM for Anomaly Detection	26
2.5.1	Autoencoders for Unsupervised Anomaly Detection	26
2.5.2	LSTM-based Sequential Modeling	27
2.5.3	Hybrid AE-LSTM Architectures	27
3	Related Work	29
3.1	Temporal Performance in ROS 2 and Its DDS Middleware	29
3.2	Security in ROS/ROS 2 and the Impact of DDS Security	30
3.3	Distribution-Shift Detection and Entropy-Based Metrics	30
3.4	Sequential Detectors with AE and LSTM	30
3.5	Evaluation for Data Streams: Windows, Adaptation, and Metrics	31
3.6	Simulation and Reproducibility	31
3.7	Gap and Specific Contribution	31
4	Methodology	32
4.1	Experimental Environment and Data Acquisition	32
4.1.1	System stack (versions).	32
4.1.2	Machine-learning stack (training).	33
4.2	Sampling Frequency and Timestamp Synchronization	34
4.3	Implementation and Properties of Entropy Metrics	36

4.3.1	Sliding Window Management	36
4.3.2	Entropies Computed on Robot Data	37
4.3.3	Comparison of Entropy Metrics	37
4.4	Other Robot-Internal Telemetry Captured by the Data-Collector Node .	38
4.5	Relationship Between Collected Metrics and Anomaly Types	39
4.6	Anomaly Generation and Extended Collection Protocol	40
4.7	Preprocessing: Dataset Cleaning and Preparation	41
4.8	Descriptive Stability of Entropy Metrics	41
4.8.1	Stationarity Verification (ADF)	42
4.8.2	Temporal Diagnosis of Entropy Metrics	43
4.9	Association Measures and Feature Selection	44
4.9.1	Correlation- and Relevance-Based Selection (8 variables)	46
4.10	Models and Training	47
4.10.1	Autoencoder	48
4.10.2	LSTM	51
4.10.3	Robust standardization and composite anomaly score	52
4.10.4	AE–LSTM Flow Summary	54
4.11	Offline Experiments: Objectives, Thresholds, and Metrics	55
4.11.1	Rationale for Offline Calibration	55
4.11.2	Sequential Offline Calibration Procedure	57
4.11.3	Description of Offline Inference	57
4.11.4	Calibration of Weights, Persistence, and Threshold	58
5	Evaluation	60
5.1	Objectives and Experimental Setup	60
5.1.1	Evaluation Metrics	60
5.2	Results	62
5.2.1	<i>Offline</i> Validation and Operating Point Selection	62
5.2.2	Event-Level Results	62
5.3	Discussion	64
5.3.1	Operating Point and Deployment Considerations	64
5.3.2	Limitations and Validity	65
5.3.3	Implications for Computational Modeling	65
5.3.4	Anomalies and Data Injection	65
5.3.5	Executive Summary	67
6	Considerations and Future Work	68
6.1	Contributions to Computational Modeling	69
6.2	Future Work	70
	References	72
	Appendices	80
A	Algorithms	80
B	Artifacts	91

1 INTRODUCTION

Autonomous mobile robotic systems based on the Robot Operating System 2 (ROS 2) operate in dynamic environments where hardware and software failures coexist with network interference and variability. Detecting anomalous behaviors early and reliably is therefore critical to preserving safety, robustness, and operational continuity. However, the multivariate and temporal nature of telemetry (sensor, control, and communication data) makes it difficult to design detectors that are both effective and lightweight.

ROS 2 was conceived as a comprehensive redesign of ROS, with a focus on data security, real-time communication, and scalability of distributed systems [66]. Nevertheless, the literature reports vulnerabilities affecting both the communication layer and software and hardware components [22], along with latency and variability challenges associated with the DDS architecture [50]. Threats such as command injection, malicious node insertion, and denial-of-service attacks [84, 18] compromise integrity, availability, and reliability. Although DDS Security introduces authentication, encryption, and access control, its adoption can bring overhead and penalties in latency and bandwidth. In critical contexts, it is therefore advisable to add complementary monitoring mechanisms that do not rely exclusively on cryptography and do not significantly increase computational cost [36].

In this work, we propose a lightweight, network-agnostic approach that operates exclusively with onboard robot telemetry. From relevant ROS 2 topics (e.g., `/cmd_vel`), arrival intervals (Δt) and kinematic magnitudes are derived. Entropy metrics are then computed over sliding windows (Shannon, transition, and KDE), following favorable evidence in networks and distributed systems [2, 35]. To capture both representational coherence and temporal dynamics, these metrics are combined with an Autoencoder-LSTM pipeline trained only on nominal data. The decision is further stabilized through a robust median/median absolute deviation (MAD) normalization and a persistence rule K that reduces false positives with low latency.

This design links a practical need (detecting relevant deviations in real time without network instrumentation) to a solution that balances sensitivity, computational cost, and traceability of decision criteria (threshold and persistence). The following sections present the specific motivation and the justification for the proposed design.

1.1 Motivation and Justification

In ROS 2 real-time scenarios, purely cryptographic mechanisms may be insufficient or too costly for monitoring system health during operation, while deep packet inspection adds complexity and overhead. Thus, complementary early-detection strategies are required that are lightweight, rely solely on onboard signals, and integrate naturally into the ROS 2/DDS ecosystem.

The proposed hybrid scheme combines: (i) entropy metrics that monitor temporal regularity and distributional shifts; (ii) an Autoencoder that provides a stable latent representation of the nominal regime and quantifies instantaneous inconsistencies via reconstruction error; and (iii) an LSTM that models one-step temporal coherence and quantifies dynamic deviations. The robust fusion based on the median and *median absolute deviation* (MAD) of both errors, together with an explicit persistence rule K , stabilizes decisions near the threshold while maintaining low response times.

Empirically, the *offline* evaluation supports this choice: Recall ≈ 0.999 is achieved while keeping the false positive rate (FPR) within the imposed constraint (FPR ≈ 0.097), together with a 100% event hit rate and zero median latency (mean ≈ 0.27 windows), evidencing timely activation consistent with real-time operation. From a practical perspective, the approach is network-agnostic, reuses existing ROS 2/DDS infrastructure, and offers a reproducible calibration procedure that facilitates transfer to different operational profiles and prevalences.

In summary, the motivation is to equip ROS 2 systems with an effective and efficient monitoring mechanism, capable of detecting relevant deviations with low latency and controlled FPR; the justification rests on the strength of the statistical component (entropy metrics), the complementarity of representation and dynamics (Autoencoder–LSTM), and the feasibility of implementation and maintenance in real environments, where early detection complements—without replacing—traditional cryptographic controls.

1.2 Problem Description

Distributed robotic systems running on ROS 2 operate under real-time constraints on DDS middleware, where failures, interference, and attacks may degrade security and performance. While authentication, encryption, and access control strengthen confidentiality and integrity, their adoption can introduce overhead and latency variability. In this context, complementary monitoring strategies are required that are lightweight, network-agnostic, and suitable for online deployment, in order to detect operational deviations with timeliness and bounded computational cost.

We address the early, robust, and low-latency detection of anomalous behaviors in a ROS 2 robot, including those caused by data-injection attacks, using exclusively onboard telemetry (without inspecting or instrumenting network traffic), while explicitly control-

ling the false positive rate (FPR), since in practice each false positive may imply interruptions or mission degradation. To this end, we construct an anomaly score from entropy metrics over sliding windows (Shannon, transition, and KDE), combined with a sequential Autoencoder–LSTM model trained only with the robot’s nominal data.

Formally, given a multivariate time series $\{x_t\}_{t=1}^N$ derived from ROS 2 topics (telemetry and commands), the detector D produces at each instant t a binary decision $y_t \in \{0, 1\}$ according to a score

$$S_t = \alpha \tilde{r}_t + \beta \tilde{p}_t,$$

where $\tilde{r}_t$ and $\tilde{p}_t$ are, respectively, the reconstruction errors (AE) and one-step prediction errors (LSTM), standardized in a robust manner (median/MAD). An alarm is triggered when $S_t > \tau$ for K consecutive instants (persistence rule). The operating threshold τ and the parameters (α, β, K) are calibrated to maximize recall under a per-window FPR constraint, with F_1 used as tie-breaker when needed. In our target regime, per-window latency meets real-time at 20 Hz.

In short, the goal is to design and calibrate an anomaly detection scheme, network-agnostic and based solely on onboard telemetry, that combines lightweight statistical metrics with sequential deep learning (Autoencoder–LSTM) to provide reliable real-time decisions on ROS 2/DDS, aligning sensitivity, FPR control, and deployment feasibility.

In the review presented in Chapter 2, we did not find any previous work that addresses exactly the same problem formulated in this dissertation. Existing studies on security in ROS/ROS 2 focus mainly on vulnerability analysis and protocol-level hardening, rather than on deploying an online learning-based detector directly onboard the robot [21, 22, 19, 47, 1]. Entropy-based anomaly detectors are typically designed for generic IP network traffic or large-scale distributed systems, not for DDS telemetry in mobile robots under real-time constraints [40, 62, 13, 86, 64, 35]. Similarly, time-series anomaly detection approaches based on autoencoders and LSTM networks are usually calibrated on industrial, energy, or generic cyber-physical data, with server-side computation and no explicit integration with ROS 2/DDS messaging [70, 48, 33, 59, 52, 82, 83, 39, 41, 15, 46]. To the best of our knowledge, there is no prior work that simultaneously combines (i) sliding-window entropy metrics over ROS 2 telemetry, (ii) their use as inputs to an AE–LSTM pipeline trained on nominal behavior, and (iii) the evaluation of data-injection attack detection on a physical ROS 2-based mobile robot. This gap motivates the specific research problem and objectives stated in this chapter.

1.3 Objectives

1.3.1 General Objective

To design, calibrate, and evaluate an anomaly detection system for ROS 2/DDS, network-agnostic (based only on onboard telemetry), that combines entropy metrics with an Autoencoder–LSTM pipeline, achieving $\text{Recall} \geq r^*$ with $\text{FPR} \leq f^*$ and decision latency $< \ell^*$ per window, while reporting both window-level and event-level metrics.

1.3.2 Specific Objectives

- Implement online computation of entropy metrics (Shannon, transition, and KDE) over sliding windows.
- Train an Autoencoder on nominal data and export a reproducible latent representation.
- Train a one-step LSTM on the latent sequence to model temporal coherence.
- Define a robust combination of errors (median/MAD) and a persistence rule K to stabilize decisions.
- Calibrate (α, β, τ, K) maximizing recall under an explicit FPR bound, with F_1 tie-breaking when needed.
- Evaluate *offline* with window-level metrics (Precision, Recall, F_1 , FPR) and event-level metrics (latency and coverage), also reporting prevalence and dataset size.
- Document deployment artifacts and procedures to ensure reproducibility (code, parameters, guides).
- Analyze sensitivity across operational regimes and QoS (e.g., $N, T, (\alpha, \beta), K$) and discuss limitations and future work (online adaptation, hardware validation).

1.4 Contributions

This work presents an anomaly detection scheme for robotic systems based on ROS 2/DDS that combines lightweight statistical metrics with deep learning models, operating exclusively with onboard telemetry. The main contributions are:

1. Online entropy monitoring with validated temporal properties. Sliding-window computation of three complementary metrics (Shannon, transition, and KDE), together with statistically calibrated thresholds and verification of temporal properties (stability and stationarity) to support operational use.

2. Hybrid Autoencoder–LSTM trained on nominal data with robust scoring. The Autoencoder learns a stable *latent* (low-dimensional) representation, and the LSTM models one-step dynamics. The anomaly score combines the reconstruction error (AE) and the one-step prediction error (LSTM) after robust standardization using the median and median absolute deviation (MAD), and is stabilized via a persistence rule K .
3. Reproducible calibration under explicit FPR constraint. Procedure to select (α, β, τ, K) maximizing recall subject to bounded FPR (with F_1 tie-breaking), with traceable reports at both window-level (Precision, Recall, F_1 , FPR) and event-level (latency, coverage, hit rate).
4. Native integration in ROS 2/DDS, network-agnostic. The detector consumes only internal topics (telemetry/commands) and publishes a binary alert once threshold and persistence are exceeded, facilitating online deployment and integration with GUI and logging without network instrumentation.
5. Reproducible experimental protocol and dataset. Simulation/acquisition flow (Gazebo/ROS 2), controlled anomaly generation, and structured logging enabling training, offline validation, and transparent comparison of operating points.
6. Empirical evidence of real-time performance. *Offline* evaluation showing high sensitivity with controlled FPR and timely event-level activation (zero median latency), supporting the feasibility of the scheme for online monitoring at 20 Hz.
7. Deployment artifacts and transfer guides. Publication of weights, scalars, thresholds, persistence, and metrics, together with calibration and production procedures, ensuring reproduction and adaptation to different regimes and prevalences (see Appendix B).

Overall, the proposal contributes an onboard detector, with low latency and explicit false-alarm control, that fuses entropy metrics and sequential deep learning (Autoencoder–LSTM) to strengthen the operational security of ROS 2/DDS robots.

1.5 Structure of the Work

This work is organized into six chapters. Chapter 1 presents the general introduction: recent advances in robotics and the new security challenges accompanying distributed systems, with emphasis on the ROS 2 middleware and its DDS stack. Relevant features and functionalities (publish/subscribe communication, QoS policies, real-time execution) and vulnerabilities reported in the literature are discussed, situating the research problem: early detection of anomalous behaviors in ROS 2 with low latency and low computational

cost. At the end, the objectives (general and specific) and contributions of this thesis are described.

Chapter 2 presents the concepts and background necessary to understand the proposed method: entropy metrics (Shannon, conditional entropy, and KDE-based entropy), sliding windows and temporal synchronization, as well as the Autoencoder–LSTM pipeline for modeling the nominal regime and performing one-step prediction. It also reviews aspects of ROS 2/DDS and the relevant QoS policies for system design.

Chapter 3 covers related work on ROS/ROS 2 security and anomaly detection in time series. Vulnerabilities identified by different authors, entropy-based approaches, and deep learning techniques are synthesized, with particular emphasis on the still underexplored combination of KDE entropy with AE–LSTM in robotic contexts over ROS 2.

Chapter 4 details the methodology. It describes the experimental environment (Gazebo/Turtlebot3 and containers), the acquisition and preprocessing procedure (sampling frequency, resampling, and 100-sample windows), the online computation of entropy metrics and kinematic variables, and the anomaly generation protocol (e.g., jitter in Δt , stop-and-go, bursts, and high/low frequency patterns). It presents the AE–LSTM pipeline, the anomaly score design with robust standardization (median/MAD) and persistence rule K , along with the offline calibration procedure (selection of weights/thresholds under explicit FPR constraint) and the set of evaluation metrics.

Chapter 5 gathers the evaluation of the method: offline experiments at both point and event levels, analysis of temporal properties and stationarity of the metrics (including ADF tests), comparison with baselines (AE only, LSTM only, threshold rules), and ablation studies (window size, parameters α , β , and K), as well as end-to-end latency and resource usage measurements.

Finally, Chapter 6 presents the conclusions and outlines directions for future work, highlighting limitations, practical implications for integration into ROS 2/DDS, and opportunities for extension (online deployment, generalization to real-world multi-robot scenarios, and incorporation of adaptive recalibration mechanisms).

2 THEORETICAL FOUNDATIONS

Here we establish the necessary conceptual foundations: the characteristics of ROS 2/DDS and QoS/security, the definition of anomaly and a threat model focused on data injection, the handling of sliding windows, entropy metrics, and the AE-LSTM architecture. This framework justifies the design decisions employed later on.

2.1 ROS 2: Fundamentals and QoS/Security Considerations

ROS 2 is a robotics software ecosystem that adopts a publish/subscribe paradigm built on the Data Distribution Service (DDS). Unlike ROS 1, the ROS 2 redesign delegates transport, discovery, and Quality of Service (QoS) control to DDS, exposing to applications a client layer (rcl/rmw) that maps ROS types to DDS/IDL types. This shift aims to improve timing behavior, scalability, and fault tolerance in distributed and real-time systems [50]; see Fig. 1 for an overview.

QoS policies (e.g., *Reliability*, *History/Depth*, *Deadline*, *Liveliness*, *Latency budget*) govern message loss and retransmission, buffer sizing, and time constraints. The literature has documented that end-to-end latency and jitter depend on (i) the DDS implementation used, (ii) message size and fragmentation, and (iii) the particular combination of QoS profiles selected; furthermore, using ROS 2 introduces additional overhead compared to direct DDS usage, which becomes more noticeable under high data loads [38, 50]. These considerations are critical for this work, as they shape the distribution of inter-arrival intervals Δt and the time series that feed the detector.

In DDS/RTPS, distributed discovery (SPDP/SEDP) exchanges participant and endpoint metadata, while liveliness checks activity through heartbeats and global identifiers (GUIDs). This control and metadata-exchange scaffolding influences the temporal variability observed by nodes and must be considered when profiling the nominal regime of ROS 2 signals [22].

To harden communication, DDS Security incorporates authentication (PKI and X.509 certificates), access control, and cryptography articulated via two signed policies: Governance (covering domains, topics, and protection levels) and Permissions/Participant

(defining per-operation and per-resource authorizations). Protection can be applied to the entire RTPS message or, more granularly, only to selected parts such as metadata and payload, with levels NONE/SIGN/ENCRYPT chosen according to confidentiality and integrity requirements [22] (see also Fig. 1).

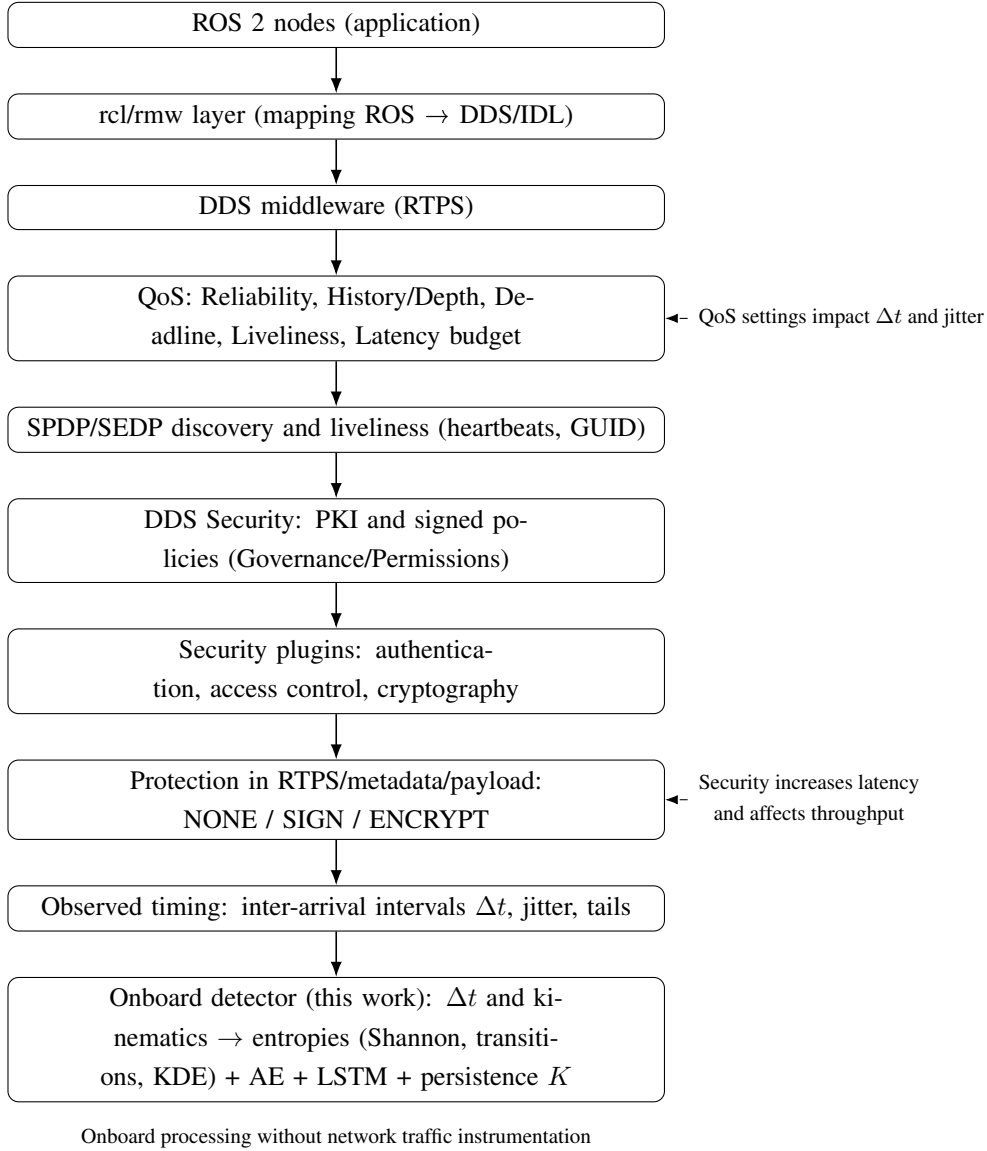


Figure 1: ROS 2 over DDS, vertical view: the client layer (rcl/rmw) maps ROS types to DDS/IDL; QoS and discovery/liveliness mechanics shape timing (inter-arrival intervals Δt , jitter). DDS Security adds authentication, access control, and encryption (NONE/SIGN/ENCRYPT) with latency/throughput impact. These signals feed the proposed detector.

Enabling signing and encryption, however, introduces measurable overhead in latency and throughput, whose magnitude depends on the DDS vendor, message size, cryptographic algorithms, and QoS configuration; significant latency increases and performance reductions relative to unsecured operation have been reported, evidencing the explicit trade-off between cryptographic guarantees and timing [22]. For time-constrained appli-

cations, selecting *what* to protect (e.g., critical payload) and *how* (e.g., SIGN vs. ENCRYPT) should be tuned jointly with QoS to preserve real-time requirements.

Even with DDS Security, the literature identifies attack surfaces and insecure configurations enabling unauthorized publish/subscribe, data injection, or denial-of-service; recent works develop formal analyses of ROS 2 communication vulnerabilities and propose verifiable security properties [84, 22]. In this context, it is justified to adopt onboard monitoring mechanisms that complement (without replacing) cryptographic controls.

This thesis targets that gap by profiling the nominal regime from onboard temporal and kinematic signals (including entropy metrics) and using an AE-LSTM scheme to detect deviations with low latency, remaining network-agnostic by not inspecting or instrumenting network traffic.

2.2 Anomaly Definition and a Data-Injection-Centered Threat Model

We define an anomaly as any observation or temporal segment that departs from the system’s nominal regime given its operating conditions [14].

Let $\{\mathbf{x}_t\}_{t=1}^N$ be the multivariate series built from telemetry and commands (e.g., inter-arrival intervals Δt , linear and angular velocities, accelerations, and speed magnitude). Let P_0 denote the nominal distribution induced by training data under regular conditions and P_1 any alternative distribution associated with faults or attacks. Detection is posed as deciding, at each instant t , between $H_0 : \mathbf{x}_t \sim P_0$ and $H_1 : \mathbf{x}_t \sim P_1$ from a score S_t built with statistical and dynamical features over sliding windows, in line with classical change-detection frameworks [6]. In practice we consider three manifestations: point-wise (an isolated out-of-distribution datum), contextual (a plausible absolute value incompatible with the operating context), and collective (an entire sequence with unusual temporal shape); this taxonomy is standard in the anomaly literature [14]. It is particularly relevant for ROS 2, where QoS policies and communication mechanics affect observed timing and, consequently, the distribution of Δt and kinematic derivatives [50, 38].

The threat model centers on data injection. We consider an adversary who attempts to alter robot behavior by publishing manipulated or spurious messages on critical topics, such as velocity commands or sensor readings. Injection may take the form of value manipulation (biases, saturations, spikes, or high/low-frequency alternations), timing manipulation (artificial jitter, bursts, and silences that modify the arrival cadence visible in the Δt series), or replay/spoofing of valid messages out of context. We assume the attacker understands DDS and can publish on specific topics, but does not simultaneously control all telemetry channels nor know the detector’s internal parameters. Unauthorized publication, malicious node insertion, and injection have been documented in ROS/ROS 2 applications, as well as formal analyses of communication vulnerabilities [21, 22, 84].

These modalities leave measurable traces in the signals. In Δt we observe increased jitter, bursts, and temporal gaps that reveal irregular cadence; in kinematic variables we observe changes in mean level, variance, and smoothness, as well as sustained coherence breaks between command (input) and state (response). Such effects motivate descriptors that characterize global and local regularity at low computational cost. We employ three complementary families of entropy metrics over sliding windows: Shannon entropy, sensitive to changes in global dispersion [74]; transition entropy (here, the entropy of inter-arrival increments), suited to capturing local cadence/jitter variations [44]; and an entropy estimated from a nonparametric density (KDE), suitable for multimodal regimes [75, 8]. Entropy-based anomaly detection has strong precedents in networks and distributed systems [40].

On this statistical basis, we integrate a hybrid model combining an Autoencoder (AE) and an LSTM network. The AE learns a stable latent representation from nominal data and its reconstruction error increases under deviations [70], while the LSTM, trained for one-step prediction in latent space, yields a prediction error that grows under dynamic deviations [48, 33]. Both errors are robustly normalized via median and MAD to reduce the impact of outliers [68] and combined into a score subject to a persistence rule K to stabilize the decision [6].

In summary, we define anomalies as deviations from a learnable nominal regime derived from telemetry and commands; we model data injection as value and/or timing alterations in critical topics; and we exploit their predictable effects on Δt and kinematic magnitudes using entropy metrics and an AE–LSTM scheme, with the goal of issuing reliable real-time decisions without instrumenting network traffic.

2.3 Sliding-Window Handling

To compute entropy metrics that are representative of the system’s recent state, the collector node applies a sliding-window scheme to the inter-arrival series $\{\Delta t_t\}$, a standard approach in stream analysis and online statistics maintenance [17, 26]. For each instant t , we define

$$W_t = \{\Delta t_{t-W+1}, \dots, \Delta t_t\}, \quad W = 100,$$

with update step $s = 1$: the most recent observation is added and the oldest one is discarded, keeping the sample size constant. On each W_t we compute Shannon entropy, transition entropy (the entropy of increments $\Delta t_t - \Delta t_{t-1}$ within W_t), and a KDE-based estimate of the differential entropy. The update and inference procedure is summarized in Algorithm 1 in Appendix A.

This scheme ensures that the analysis faithfully reflects the most recent conditions,

preventing stale measurements from distorting detection in dynamic contexts [26]. The choice $W = 100$ is a practical compromise between sensitivity (ability to react to changes) and stability (estimator variance). During the warm-up phase, until W samples are available, the descriptors are computed but no decisions are issued.

From a computational standpoint, count-based metrics (discrete histograms for Shannon entropy and for the increments) admit $O(1)$ updates by maintaining bin counters when a sample enters or leaves the window [17]. For the KDE-based entropy, we fit a Gaussian KDE on the last $m = \min(W, |V|)$ samples, with bandwidth h (Scott’s or Silverman’s rule), and use a leave-one-out plug-in estimator of the differential entropy [80, 75].

To justify per-window normalizations and thresholds, we assume quasi-stationarity within each W_t . This hypothesis was verified via the Augmented Dickey–Fuller (ADF) test, checking for unit roots under no-constant, constant, and constant-plus-trend specifications [20, 69]. The number of lags k was selected using AIC/BIC within Schwert’s practical bound for $k_{\max}$ [72]. Decisions were made by comparing to critical values or using p -values. The results, summarized in Table 5 in Section 4.8.1, support the use of robust standardization (median/MAD) and fixed per-window thresholds under nominal operation [68].

2.4 Entropy Metrics

Building on the sliding-window scheme introduced in Section 2.3, this section details the statistical entropy-based descriptors that constitute the first stage of our detector. We formalize three entropy descriptors computed on each window W_t : (i) Shannon entropy of the inter-arrival times, (ii) transition entropy of consecutive increments, and (iii) a KDE-based differential entropy. These quantities act as lightweight online indicators of distributional change in ROS 2/DDS telemetry.

2.4.1 Shannon Entropy

Shannon entropy [74], proposed in 1948 within information theory, measures the uncertainty or information content associated with a random variable. It is widely used to analyze distributions, as it summarizes the degree of randomness or disorder in the data.

Let X be a random variable that can take values in a finite set $\mathcal{X}$. We denote by $p_X(x)$ the probability (or empirical frequency) of each value $x \in \mathcal{X}$. Entropy is defined as

$$H(X) = - \sum_{x \in \mathcal{X}} p_X(x) \log_2 p_X(x), \quad (1)$$

where the base-2 logarithm expresses information in bits. In practice, when observations are continuous, p_X can be estimated with histograms or via kernel density estimation (KDE) [25, 73, 75]. In our implementation, since the window W_t is treated as a discrete

sequence V , we use the empirical probability mass function obtained from frequency counts within the window (one bin per distinct value); KDE is reserved for the continuous entropy descriptor introduced in this section. To compare across windows with K distinct values, the normalized entropy $H/\log_2 K$ may be used; for very short windows, a Miller–Madow bias correction is optional.

High entropy indicates more uniform distributions (greater uncertainty and lower predictability), whereas low entropy reflects concentration on a few outcomes (lower uncertainty). In anomaly detection, appreciable changes in the probability distribution often reveal unexpected events. In ROS 2 systems, inter-node messaging exhibits nominal temporal and structural patterns; variations in publication dynamics captured, for example, by the entropy of Δt over sliding windows can signal communication faults, malicious alterations, or unforeseen behaviors. This idea has been successfully applied in network traffic and distributed systems: Radivilova et al. [62] use sliding windows to identify anomalous patterns; Callegari et al. [13] and Ziviani et al. [86] show entropy’s sensitivity to deviations in traffic structure; and recent works [35, 4] reinforce its utility as a robust descriptor, motivating its adoption in robotic contexts over ROS 2. In our detector, the entropy of Δt is computed on each window W_t and monitored as a time series: we apply robust standardization (median/MAD) under nominal conditions and raise a local entropy alert when the standardized value exceeds a calibrated threshold. These entropy-based alerts are then combined with the autoencoder–LSTM score and a persistence rule (requiring K consecutive exceedances) in the final decision logic.

2.4.2 Transition Entropy

Here we present *transition entropy* to refer to Shannon entropy applied not to the absolute values of a time series but to its consecutive sample increments. This captures the uncertainty associated with local changes and is suitable for characterizing cadence variations (jitter, bursts, silences) in robotic and real-time communication systems, without incurring the cost of modeling full joint or conditional distributions.

Let $X = \{x_1, \dots, x_n\}$ be a one-dimensional sequence (e.g., inter-arrival times Δt or a kinematic magnitude). Define the increment series:

$$\Delta X = \{\Delta x_1, \dots, \Delta x_{n-1}\}, \quad \text{with} \quad \Delta x_i = x_{i+1} - x_i, \quad (2)$$

and, in a sliding window W_t , treat the corresponding increment sequence as a discrete sequence V . Let $\{p_k\}_{k=1}^K$ denote the empirical probabilities of the distinct increments in V , obtained from frequency counts (one bin per distinct value). The transition entropy is then

$$H_{\text{tra}}(W_t) = - \sum_{k=1}^K p_k \log_2 p_k, \quad (3)$$

as implemented in Algorithm 3 in Appendix A.

Interpretively, more dispersed increments (jitter spikes, bursts, or gaps) broaden the distribution of Δx and tend to increase H_{tra} , whereas rigid change patterns reduce it. This approach differs from Bandt–Pompe permutation entropy, which relies on ordinal patterns and ignores variation magnitude [5]; here the measure is explicitly sensitive to change magnitudes, making it complementary to ordinal metrics and especially informative under cadence alterations. Conceptually, it is related to entropy-of-difference (increment signs) and increment entropy (sign–magnitude coding), used as complexity/detection measures in time series [53, 44]; it is also related to ordinal-transition-based change-detection approaches [78].

Operationally, the transition entropy used here (Shannon over the empirical distribution of Δx_t within windows) differs from computing Shannon/KDE over x_t , from ordinal permutation entropy, and from standard Increment Entropy (with $m > 1$) due to its focus on change magnitude and its marginal (wordless) nature. It only coincides with earlier metrics under degenerate choices, e.g., $\Delta x_t \in \{-, 0, +\} \Rightarrow$ entropy of difference; or sign–magnitude symbolization with $m = 1 \Rightarrow$ one-dimensional Increment Entropy. Otherwise, the metrics are not identical but complementary.

In ROS 2, we apply H_{tra} over sliding windows of Δt and discrete derivatives of kinematic variables to capture local irregularities in publication and robot dynamics, yielding a computationally light descriptor sensitive to perturbations relevant for early detection.

2.4.3 KDE-Based Entropy

Beyond entropy measures computed over discretized distributions, we employ a continuous variant based on Kernel Density Estimation (KDE), which provides a nonparametric, smoothed approximation to the probability density function (PDF) [75, 8]. Given samples $\{x_1, \dots, x_n\}$, the univariate KDE is

$$\hat{f}(x) = \frac{1}{n h} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right), \quad (4)$$

where n is the number of samples in the window, K is a normalized kernel ($\int K(u) du = 1$, e.g., Gaussian), and $h > 0$ is the bandwidth controlling smoothness. Selecting h is critical; in practice one uses general-purpose rules (Silverman/Scott) or automatic methods such as Sheather–Jones via validation [75, 73].

With this density, the associated differential entropy is

$$H_{\text{KDE}} = - \int \hat{f}(x) \log \hat{f}(x) dx, \quad (5)$$

with logarithm in base e (nats) or base 2 (bits). In applications, it is evaluated using a

leave-one-out Monte Carlo-style estimator to reduce bias:

$$\hat{H}_{\text{KDE}} \approx -\frac{1}{n} \sum_{i=1}^n \log \hat{f}_{-i}(x_i), \quad (6)$$

where $\hat{f}_{-i}$ is the KDE excluding x_i . When the support is semi-infinite (e.g., $x = \Delta t \geq 0$), a logarithmic transform or boundary corrections help avoid bias from mass leakage outside the domain [75].

KDE-based entropy has been used to analyze time series and detect changes or atypical behaviors by offering a robust measure of global uncertainty without relying on a fixed discretization [8]. Recent examples explore multiscale variants and functional extensions for complex data, as well as biomedical applications [43, 27]. Here, we compute H_{KDE} over sliding windows of temporal signals (e.g., Δt and kinematic magnitudes) to complement count- and increment-based entropy measures, increasing sensitivity to regime shifts with low modeling cost.

2.5 Autoencoder + LSTM for Anomaly Detection

2.5.1 Autoencoders for Unsupervised Anomaly Detection

Autoencoders (AEs) are neural networks trained to reproduce their inputs at the output through a constrained hidden representation [31, 28, 9]. Given an input vector $\mathbf{x} \in \mathbb{R}^d$, an encoder network f_θ maps $\mathbf{x}$ to a latent code $\mathbf{z} = f_\theta(\mathbf{x})$, while a decoder g_ϕ reconstructs $\hat{\mathbf{x}} = g_\phi(\mathbf{z})$. Training typically minimizes a reconstruction loss, such as the mean squared error $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$, over a dataset assumed to represent nominal (i.e., non-anomalous) behaviour. Under suitable capacity and regularization, the AE learns to represent regular patterns of variation while assigning larger reconstruction errors to inputs that deviate from the training distribution.

For anomaly detection, the reconstruction error $e^{\text{AE}} = \|\mathbf{x} - \hat{\mathbf{x}}\|$ is interpreted as an anomaly score: samples that are poorly reconstructed relative to the nominal baseline are flagged as atypical [70, 12]. A rich body of work has explored regularized and robust AEs—including denoising, sparse and contractive variants—to prevent trivial identity mappings and encourage latent representations that capture stable structure rather than noise or idiosyncrasies [79, 55, 65]. Recent surveys review autoencoder families and their applications in machine learning and anomaly detection across multiple domains [10, 14, 12]. In networked and robotic systems, AE-based detectors have been successfully employed to capture deviations in latent representations of telemetry and traffic features [76, 15].

2.5.2 LSTM-based Sequential Modeling

While AEs provide a static snapshot of deviation at the level of individual observations, many cyber–physical systems exhibit strong temporal dependencies. Recurrent neural networks (RNNs) address this by maintaining a hidden state that evolves over time, but standard RNNs suffer from vanishing and exploding gradients when modelling long-range dependencies. Long Short-Term Memory (LSTM) networks [32] mitigate this limitation by incorporating gating mechanisms (input, output and forget gates) that regulate information flow, enabling the network to retain and update relevant temporal information over longer horizons.

In time-series anomaly detection, a common strategy is to train an LSTM to perform one-step prediction of a sequence that reflects nominal dynamics. Given a latent or observable sequence $\{y_t\}$, the LSTM learns to predict $\hat{y}_{t+1}$ from the past, and the prediction error $e_t^{\text{LSTM}} = \|y_{t+1} - \hat{y}_{t+1}\|$ is used as an anomaly score: sustained increases in prediction error indicate departures from the learned dynamics, such as regime changes, persistent jitter or unexpected stop–go patterns [48, 49]. LSTM-based detectors have shown good performance in high-dimensional temporal streams such as spacecraft telemetry and industrial sensors [33, 12], and are widely used in streaming anomaly detection surveys [45].

2.5.3 Hybrid AE–LSTM Architectures

Hybrid architectures that combine AEs and LSTMs seek to exploit the complementary strengths of static reconstruction and temporal prediction. A common design encodes high-dimensional inputs into a latent representation with an AE, and then models the temporal evolution of this latent code with an LSTM [70, 49]. The AE reduces noise and redundancy while capturing salient structure in a compact or regularized latent space; the LSTM focuses on learning temporal coherences and dynamics in that space. Anomaly scores can then be derived from reconstruction error, prediction error, or a combination of both, enhancing sensitivity to both structural and dynamical deviations [33, 59, 39, 41, 42, 3, 52, 82, 83].

Within this general framework, robust statistics (e.g., median- and MAD-based normalization) and simple thresholding rules are often employed to turn continuous scores into binary decisions, optionally with persistence requirements to dampen spurious spikes and enforce temporal consistency in alarms [68, 6, 29, 54, 56, 34, 85, 77]. Building on these theoretical foundations, the present work specifies and evaluates a concrete AE–LSTM architecture tailored to ROS 2/DDS telemetry and entropy-based descriptors in the Methodology chapter, where the exact model configuration, training procedure and decision rule are detailed.

On this conceptual basis, Chapter 4 details the proposed pipeline (entropies + AE–LSTM) and, later, Section 5.3.4 discusses how detected deviations correspond to data-

injection patterns.

3 RELATED WORK

The goal of this chapter is to situate our work within the broader landscape of anomaly detection for ROS 2/DDS-based robotic systems. We first motivate why temporal performance, security considerations, and distribution shifts matter in practice for real-time operation. We then outline how entropy-informed signals and sequential deep models can complement each other to detect meaningful deviations under tight latency and resource constraints. Finally, we clarify the evaluation perspective adopted here—one that emphasizes event-level metrics, latency, and reproducibility—so that subsequent methodological choices are transparent and well grounded.

To guide the reader, the literature reviewed is organized along six axes: (i) temporal performance of ROS 2 and its DDS middleware; (ii) security in ROS/ROS 2 and the impact of DDS Security; (iii) distribution-shift detection and entropy-based metrics; (iv) sequential detectors with autoencoders and LSTM models; (v) evaluation protocols for data streams and time series; and (vi) simulation and reproducibility. This structure synthesizes findings that motivate the design and evaluation choices adopted in this thesis.

3.1 Temporal Performance in ROS 2 and Its DDS Middleware

Early ROS 2 performance studies report that end-to-end latency and jitter are highly sensitive to the DDS implementation, to QoS policies (e.g., Reliability, History/Depth), and to message size/fragmentation. Maruyama *et al.* characterize these dependencies and compare representative configurations in controlled scenarios with emphasis on the communications stack and its temporal implications [50]. Kronauer *et al.* extend the analysis in multi-node systems, quantifying the overhead introduced by the ROS 2 layer with respect to direct DDS use and highlighting the role of QoS in aggregate temporal behavior [38]. To instrument and diagnose real systems, Bédard *et al.* introduce `ros2_tracing`, a low-overhead tracing framework that enables causal inspection and fine-grained measurement of delays and queueing at runtime [7]. These results motivate, in our work, the decision to fix a default configuration and to control the false positive rate explicitly under timing constraints.

3.2 Security in ROS/ROS 2 and the Impact of DDS Security

In classic ROS, Dieber *et al.* systematize the attack surface (spoofing, man-in-the-middle, denial of service) and discuss architecture- and channel-level countermeasures [21]. For ROS 2, DiLuoffo *et al.* argue for a holistic approach integrating DDS configuration, QoS policies, and DDS Security, pointing out security–performance trade-offs [22]. Recent studies of vulnerabilities and misconfigurations in ROS 2 underscore that enabling DDS Security can introduce overhead and side effects that affect real-time operation if not carefully managed [84]. In parallel, applied work explores machine-learning-based intrusion detection in ROS ecosystems, validating the viability of supervised approaches in simulation [4].

3.3 Distribution-Shift Detection and Entropy-Based Metrics

Distribution-shift detection has proven effective in networking and cyber-physical systems: Lakhina *et al.* show that changes in traffic-feature distributions expose structural anomalies not visible from point values [40]. Building on this, our approach uses sliding-window entropies to capture changes in cadence and dispersion. We rely on classical binning rules (Freedman–Diaconis, Scott) [25, 73] and kernel density estimation (KDE) [75]. KDE-based entropies have been reported for time-series contexts [43, 27], while Increment Entropy—which symbolically encodes sign and magnitude of increments—offers sensitivity to local changes [44]. Since heavy tails and outliers can degrade naive thresholds, we adopt robust standardization based on median and MAD [68].

3.4 Sequential Detectors with AE and LSTM

Autoencoders (AE) provide low-dimensional representations of nominal regimes and are widely used for reconstruction-based detection [31, 28, 79, 55, 65]. LSTMs capture medium- and long-range temporal dependencies [32]. In time-series anomaly detection, AE and LSTM have been applied both separately and jointly: AEs for reconstruction with thresholds [70], LSTMs for sequence modeling and prediction [48], and hybrid variants (e.g., VAE–LSTM, AE–LSTM) in industrial and spatiotemporal domains [33, 59, 41, 39], and have demonstrated high sensitivity even with unlabeled data. Our pipeline follows this line, combining an entropy view (marginal change), an AE (representation consistency), and an LSTM (sequential coherence), with thresholding and temporal persistence to control latency and false positives in operation.

3.5 Evaluation for Data Streams: Windows, Adaptation, and Metrics

The online nature of the task motivates sliding windows and adaptation mechanisms: Datar *et al.* study maintaining statistics under sliding windows [17], and Bifet & Gavalda propose ADWIN, an adaptive window scheme sensitive to drift [11]. To analyze stationarity and define stable thresholds, we rely on classical ADF tests [20, 69, 72]. For evaluation, beyond confusion-matrix point metrics, ROC curves and TPR/FPR are used [24], together with event-oriented metrics tailored to range anomalies (coverage, event hit rate, activation latency) [77]. In this work we adopt both pointwise and event-level views to reflect the detector’s practical utility under timing constraints.

3.6 Simulation and Reproducibility

Simulation in Gazebo allows controlled variable settings, reproducible traces, and the construction of a nominal baseline prior to hardware deployment [37]. Combined with containers and default ROS 2/DDS configurations, this strategy reduces experimental variance attributable to the middleware and isolates the contribution of the detector and its input metrics. To enhance repeatability, simulations were run in Docker on ROS 2 Humble with Fast DDS and Gazebo Classic, and the anomaly generator used a fixed seed (`random.seed(42)`).

3.7 Gap and Specific Contribution

Although there are advances in entropic metrics, AE- or LSTM-based detectors used in isolation, and evaluations oriented toward overall accuracy, the literature lacks: (i) a *native, lightweight* integration of complementary entropies (Shannon, transitions, and KDE) with an *AE-LSTM* pipeline trained only on nominal data; (ii) robust normalization (median/MAD) and a *persistence rule* with parameter K to stabilize decisions; and (iii) *reproducible calibration* with an *explicit FPR constraint* and *event-level reporting* under ROS 2/DDS temporal requirements. This thesis fills that gap with an onboard, network-agnostic detector that runs online under ROS 2/DDS timing constraints, without inspecting or instrumenting network traffic, and that is readily integrated into ROS 2.

4 METHODOLOGY

We describe the experimental environment (Gazebo/ROS 2 Humble with Fast DDS), the data acquisition pipeline, preprocessing, and the online computation of entropy-based metrics and kinematic variables. We detail the AE-LSTM pipeline, the construction of the anomaly score with robust normalization and a persistence rule with parameter K , and the calibration procedure under an explicit false positive rate (FPR) constraint.

The proposed methodology proceeds in two main stages. First, we establish a nominal behavioral baseline via controlled simulations in an empty Gazebo world, which enables training a hybrid deep-learning detector. Second, we analyze new data under mixed conditions (nominal and anomalous) to assess the system’s ability to detect meaningful deviations in the robot’s operating dynamics.

4.1 Experimental Environment and Data Acquisition

All experiments were conducted on ROS 2 Humble with Fast DDS using out-of-the-box settings—no XML profiles, DDS Security disabled, and default QoS—inside a Docker container. Simulation ran in Gazebo 11.10.2 (Classic) with a *TurtleBot3* in an empty world to isolate the nominal regime from external interference [37]. This baseline is common in ROS 2 performance and security evaluations: studies first characterize latency/jitter with defaults and then examine the impact of QoS and DDS Security given their temporal overhead [50, 38, 36, 22].

4.1.1 System stack (versions).

- **Base OS and container.** Ubuntu 22.04.4 LTS (Jammy) inside Docker 27.1.2 (build d01f264).
- **ROS 2 and DDS.**
ROS 2 Humble (`$ROS_DISTRO=humble`) with Fast DDS (`rmw_fastrtps_cpp`).
- **Fast DDS packages installed via APT.**
 - `ros-humble-fastrtps (2.6.8-1jammy)`

- `ros-humble-rmw-fastrtps-cpp` (6.2.6-1jammy)
 - `ros-humble-rmw-fastrtps-shared-cpp` (6.2.6-1jammy)
 - `ros-humble-rosidl-`
`typesupport-fastrtps-{c,cpp}`
(2.2.2-2jammy)
- **ROS 2 metapackages.** `ros-core` (0.10.0-1jammy.20240524), `ros-base` (0.10.0-1jammy.20240530) y `desktop` (0.10.0-1jammy.20240617).
 - **QoS.** No QoS profiles were manually tuned: the sensor topics (`/laser_scan`, `/imu/data`, `/odom`) used the *sensor_data* profile (Volatile, Best Effort, low *depth*), whereas `/cmd_vel` and `/movement_label` inherited *default/system_default* (Reliable, KeepLast, typical *depth* of 10), following the official ROS 2 QoS documentation and the `rclcpp` QoS classes [58, 67]. These default values were sufficient for our scenarios and helped avoid additional latency due to *acknowledgments* or deep queues [50, 38].

4.1.2 Machine-learning stack (training).

- **Python environment.** Python 3.11.2 with NumPy 2.2.5, Pandas 2.2.3, PyTorch 2.6.0 (CPU only), `scikit-learn` 1.6.1, `Joblib` 1.4.2, and `Matplotlib` 3.10.1.
- **Traceability.** A machine-readable log of the exact package versions is included in `training_env_versions.json`.
- **Random seeds.** Unless explicitly stated otherwise, no seeds were fixed during training; only the anomaly generator used a fixed seed (42) to reproduce the injection patterns.

We purposely ran ROS 2 Humble + Fast DDS with factory configuration (default QoS, DDS Security off) to reduce middleware confounders and isolate the detector’s effect on onboard signals. The literature shows ROS 2 timing strongly depends on DDS stack, QoS, and message size/fragmentation; any tuning shifts latency/jitter and complicates causal attribution [50, 38]. Enabling DDS Security adds measurable overhead and PKI/policy decisions that increase experimental variance [22, 36], and misconfigurations can introduce risks or side effects [84, 18]. Moreover, tracing tools indicate the system already exhibits sufficient timing variability without further tuning [7]. Fixing defaults thus provides a reproducible, honest baseline; sensitivity to QoS/DDS Security is left for future work.

Robot actions were generated by a motion-simulator node that published velocity commands on `/cmd_vel`, executing controlled random trajectories (without injecting

artificial anomalies), including straight segments, gentle curves, turns, and zig-zag sequences. It also published descriptive labels on `/movement_label` for each motion type, enabling a labeled dataset under nominal conditions.

A data-collector node continuously recorded onboard telemetry during simulation in real time. Stage 1 in Fig. 2 corresponds to node start-up inside the ROS 2 environment, including initialization of the CSV file and internal buffers for inter-arrival times and entropy metrics. In Stage 2, the node subscribes to the main telemetry and labeling topics: velocity commands (`/cmd_vel`), odometry (`/odom`), LiDAR readings (`/laser_scan`), inertial measurements (`/imu/data`), motion labels (`/movement_label`), and the injected-attack label (`/attack_type`).

When a new message arrives (Stage 3), primarily from `/cmd_vel`, the node updates the inter-arrival series $\{\Delta t_t\}$ using the wall-clock time. On the current sliding window ($W=100$ samples), it then computes three entropy metrics (Stage 4): Shannon entropy, transition entropy (on first-order differences), and kernel-density-based entropy (KDE), as formalized in Section 2.4.

Stage 5 aggregates and processes additional sensor information from the latest samples of `/cmd_vel`, `/odom`, `/laser_scan`, and `/imu/data`. The node extracts linear and angular velocities, odometry pose, LiDAR minimum/maximum ranges, IMU accelerations, and derived kinematic quantities such as linear/angular acceleration, velocity magnitude, and jerk.

In Stage 6, all features for the current time step (timestamp, kinematic variables, laser/IMU attributes, the three entropy values, and motion/attack labels) are appended as a single row to a structured CSV file at an effective rate of about 10 Hz, which is sufficient to characterize the robot’s dynamics. During offline processing, the time series are synchronized and resampled to 20 Hz (linear interpolation for short gaps) for analysis and model consumption. Finally, Stage 7 publishes informational log lines containing the entropy metrics, the motion label, the attack type, and an auxiliary KDE-based anomaly flag. This flag is used only as a diagnostic aid during data collection and does not participate in the AE-LSTM detection pipeline trained on the CSV.

Running everything inside a single container ensured time synchronization and environmental consistency. Figure 2 summarizes this workflow, enumerating each stage described above.

4.2 Sampling Frequency and Timestamp Synchronization

The sampling frequency adopted for data collection was 10 Hz, a compromise between temporal resolution and computational efficiency. This decision was guided by two complementary aspects: on the one hand, the need to capture dynamic events of the robot—such as turns, braking, or abrupt transitions—with sufficient granularity; on the

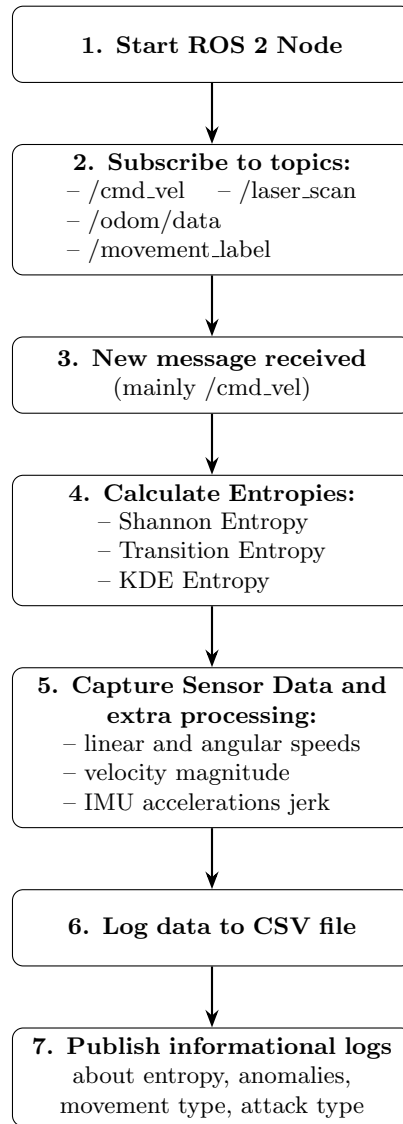


Figure 2: Data-collector node workflow: from ROS 2 subscriptions to entropy computation, feature extraction, CSV logging, and informational logs.

other hand, the convenience of keeping the data rate manageable, avoiding system overload or burdensome post-processing.

Prior studies have demonstrated that frequencies between 5 and 20 Hz are effective for anomaly detection in robotic systems [15, 46, 61]. For example, [15] employed 10 Hz to record anomalous events in manipulators during more than five hours of continuous operation, successfully classifying human collisions and manipulation errors. Similarly, [46] reported that up to 10 Hz was adequate for real-time detection without compromising system performance. Finally, [61] evaluated forecasting models at 10 Hz, concluding that this frequency offered a balanced trade-off between sensitivity and stability.

Based on this evidence, adopting a 10 Hz frequency was deemed to ensure effective coverage of relevant events, aligning with the specialized literature and maintaining com-

putational feasibility in both simulated and real scenarios.

Although acquisition was performed at 10 Hz, for modeling and analysis all signals were synchronized and resampled to 20 Hz (using linear interpolation for short gaps). This ensured uniform sliding windows of 100 samples (≈ 5 s), aligned topics subject to mild jitter, and simplified preprocessing by maintaining consistent lengths per window/epoch. This resampling is applied only during feature engineering and does not alter labels or the temporal order of the records.

4.3 Implementation and Properties of Entropy Metrics

Robot-internal data were analyzed through the real-time computation of three entropy metrics: Shannon entropy, transition entropy (based on successive differences), and kernel-density-based entropy (KDE). These metrics were computed over sliding windows of 100 samples, maintaining an updated and sensitive representation of the system's recent behavior. In the implemented node, the entropy calculations are applied exclusively to the series of message inter-arrival intervals Δt derived from the `/cmd_vel` topic, which reflects the cadence of velocity commands.

The implementation is organized into two modules: (i) an *entropy module* that maintains the sliding window of Δt for `/cmd_vel` and updates the three entropy descriptors online; and (ii) a *telemetry module* that records the raw values published on relevant topics (velocity commands, odometry, LiDAR ranges, IMU accelerations, and motion/attack labels) without applying additional entropy transformations to those signals. All quantities (entropies and telemetry) are written to a structured CSV file for subsequent offline synchronization, resampling, and modeling.

4.3.1 Sliding Window Management

To compute representative entropy metrics adaptively to recent system behavior, the data-collector node implements a sliding-window scheme over the series of message inter-arrival times Δt .

The window maintains the latest $W = 100$ observations of Δt , on which Shannon, transition, and KDE-based entropies are calculated. Upon each new message, the system updates the window by adding the new interval and discarding, if needed, the oldest value, keeping the sample size constant. The detailed update and computation procedure is described in Algorithm 1 in Appendix A.

This approach allows the statistical analysis to reflect the most recent system conditions, avoiding distortions from outdated measurements in dynamic contexts. The choice of $W = 100$ samples represents a practical trade-off between sensitivity (responsiveness to changes) and stability (reduced variance of the estimators).

4.3.2 Entropies Computed on Robot Data

4.3.2.1 Shannon Entropy

Shannon entropy is calculated from the distribution of inter-arrival times Δt of messages received on subscribed topics. The procedure begins with frequency counts of Δt values, from which relative probabilities are estimated. With these probabilities, the classical entropy formula $H = -\sum p_i \log_2 p_i$ is applied, including a numerical correction term to avoid evaluating $\log(0)$ and to ensure computational stability. The step-by-step description of this process is given in Algorithm 2 in Appendix A.

4.3.2.2 Transition Entropy

This metric is based on the analysis of differences between consecutive values in the Δt series, capturing the uncertainty associated with changes in message arrival cadence. The procedure consists of generating a new series from pairwise differences of consecutive values, and then applying Shannon entropy on this series to estimate transition variability. A detailed description is provided in Algorithm 3 in Appendix A.

4.3.2.3 Kernel-Density-Based Entropy (KDE)

KDE entropy provides a continuous representation of the underlying data distribution and is less sensitive to discretization choices. In our case, a Gaussian kernel density estimator $\hat{f}_h$ is fitted over the window of values, and the log-density is then evaluated at the samples themselves to approximate differential entropy using a plug-in estimator. Computationally, this involves computing $\ell_i = \log \hat{f}_h(x_i)$ and approximating $H_{\text{KDE}} \approx -\sum_i \exp(\ell_i) \ell_i$ (with optional normalization by n depending on the reporting protocol). The detailed procedure is described in Algorithm 4 in Appendix A.

4.3.3 Comparison of Entropy Metrics

Shannon entropy quantifies overall uncertainty in the distribution of inter-arrival times (Δt), offering a measure of dispersion or uniformity [74, 64]. It is useful for identifying changes in the global variability of the system.

Transition entropy analyzes differences between consecutive intervals, capturing the system's temporal dynamics. It is particularly sensitive to abrupt changes such as jitter or bursts of messages, which may not be evident in the static distribution of intervals [78].

Finally, KDE-based entropy uses continuous probability density estimation to evaluate uncertainty, enabling the detection of subtle changes in distribution shape, such as the emergence of bimodality or heavy tails [64].

The combination of these metrics provides a comprehensive view of the system's temporal behavior, enhancing anomaly-detection capability across diverse operating scenarios.

Table 1: Comparison of entropy metrics used.

Metric	What It Measures	Main Sensitivity	Ref.
Shannon entropy	Global uncertainty of the inter-arrival time distribution.	Changes in dispersion and uniformity of arrival times.	[74, 64]
Transition entropy	Variability of transitions between consecutive intervals.	Abrupt changes in temporal dynamics (jitter, bursts, local disorder).	[78]
KDE-based entropy	Uncertainty estimated continuously via a non-parametric probability density.	Subtle changes in distribution shape (tails, multimodality).	[43]

4.4 Other Robot-Internal Telemetry Captured by the Data-Collector Node

In addition to entropy metrics computed over message inter-arrival times, the data-collector node also acquires a set of complementary features from different ROS 2 topics, with the goal of building a more comprehensive representation of the robot’s operational state.

Among the recorded variables are linear and angular velocities extracted from the `/cmd_vel` topic, which directly reflect the movement commands issued to the robot. From these velocities, linear and angular accelerations are computed as temporal derivatives, together with the velocity magnitude, defined as the scalar combination of linear and angular components. Furthermore, jerk is estimated as the derivative of linear acceleration, enabling the detection of abrupt changes in motion dynamics.

The system also records odometry information from the `/odom` topic, capturing the robot’s position in the plane (x, y) and orientation (θ) . In parallel, minimum and maximum distance measurements obtained from the laser scanner (`/laser_scan`) are collected, offering an indirect measure of proximity to obstacles or environmental changes.

Additionally, acceleration values along the three axes (x, y, z) are acquired from the inertial measurement unit (IMU) via the `/imu/data` topic. These data enrich the robot’s kinematic information, allowing evaluation of internal forces and external disturbances affecting the system.

All data are stored synchronously with timestamps, allowing subsequent analysis both in isolation and in correlation. This multimodal acquisition strategy provides a solid basis for characterizing normal and anomalous behaviors, and is fundamental for building more accurate and robust anomaly-detection models.

Table 2 summarizes the main variables captured, their acquisition sources, and their role within the system.

Table 2: Variables captured by the data collector node.

Category	Variable(s)	Data Source
Velocity	Linear velocity (v), angular velocity (ω)	<code>/cmd_vel</code>
Acceleration	Linear acceleration, angular acceleration	Internally computed
Velocity magnitude	Combined magnitude of v and ω	Internally computed
Jerk	Derivative of linear acceleration	Internally computed
Odometry	Position (x, y), orientation (θ)	<code>/odom</code>
LiDAR	Minimum and maximum detected distances	<code>/laser_scan</code>
IMU	Accelerations in x, y, z	<code>/imu/data</code>
Labels	Movement type, attack type	<code>/movement_label</code> , <code>/attack_type</code>
Entropies	Shannon, transition, KDE	Internally computed

4.5 Relationship Between Collected Metrics and Anomaly Types

The set of metrics employed in this work was deliberately designed to capture distinct manifestations of anomalies in robotic and communication systems based on ROS 2. Each metric contributes sensitivity to specific failures, atypical behaviors, or attacks, so that collectively they enable more robust and specific detection of irregularities.

Table 3 summarizes the relationship between the metrics considered, the type of anomaly they help identify, and a brief justification of each association.

Table 3: Relationship between collected metrics and possible anomalies detected.

Metric	Anomaly Type	Justification
Message inter-arrival time (Δt)	Jitter, bursty publication	Anomalous variations in topic publication intervals may indicate desynchronization, node overload, or burst injection.
Shannon entropy	Irregularities in internal transmission	Changes in the distribution of published intervals/values suggest loss of regularity or manipulation of message flow.
Transition entropy (ordinal patterns)	Abrupt dynamic changes	Detects alterations in the relative sequence of consecutive intervals/values, sensitive to jitter and short bursts.

Metric	Anomaly Type	Justification
KDE-based entropy	High dispersion or collapse of variability	Reveals modifications in distribution shape (tails, multimodality), useful for identifying regime changes in the system.
Linear and angular velocity	Command injection, control/actuator failures	Significant deviations from the normal motion profile may indicate anomalous commands or mechanical degradation.
Linear and angular acceleration	Impacts, unexpected sharp maneuvers	Unusual accelerations reveal collisions, slips, or unauthorized control inputs.
Velocity magnitude and <i>jerk</i>	Dynamic motion anomalies	Sudden changes in magnitude or acceleration derivative help detect control failures or saturation.
LiDAR (min/max)	Unexpected obstacles, environmental changes	Atypical variations in minimum/maximum ranges indicate physical changes in the environment or local interference.
IMU (x, y, z accelerations)	Vibrations, falls, impacts	Anomalous IMU measurements expose structural risks or unexpected excitations.
Movement/attack labels	Validation and <i>ground truth</i>	Enable comparison of automatic detection with known labels for evaluation and threshold tuning.

The integration of multiple metrics associated with both message dynamics and robot physical behavior allows a more robust anomaly-detection approach sensitive to diverse irregularities. Each metric captures complementary aspects of normal operation and possible disruptions, facilitating early identification of failures, attacks, or unexpected behaviors. The joint use of entropy-based analysis, kinematic variables, and sensor readings thus provides a holistic view of the system’s operational state, forming a solid basis for subsequent evaluation under controlled and real anomalous scenarios.

The methodology presented establishes a data acquisition and analysis system oriented toward early anomaly detection in robotic systems based on ROS 2. Through the implementation of metrics grounded in entropy analysis, motion dynamics, and inter-node communication characteristics, it aims to capture early indicators of irregular behaviors. The design of adaptive thresholds and the selection of relevant variables enable a flexible and robust approach to dynamic scenarios and potential security threats.

4.6 Anomaly Generation and Extended Collection Protocol

The motion-simulator node includes an `anomaly_mode` parameter. In the configuration used, anomalies are assigned relatively long durations (5–10 s) to increase the

proportion of anomalous samples and provide richer labeled data for model fitting. Anomalies are activated with a dynamic probability that targets an anomalous time fraction of approximately 15–20% of the total runtime. Implemented patterns include: (i) high-frequency command sequences, (ii) low-frequency patterns with large inter-message gaps, (iii) irregular stop-and-go behavior alternating between motion and standstill, and (iv) intermittent bursts of repetitive commands followed by random pauses. Each anomaly instance runs for a random duration within the 5–10 s range. Between anomaly intervals, the node generates nominal motions (straight segments, smooth curves, spins, zigzag paths, and obstacle-avoidance maneuvers). All actions are labeled online on `/movement_label`, enabling the association of sensor signals with their behavior class. Additionally, the node maintains a persistent anomaly log (CSV) that records start/end timestamps and the active anomaly type.

To support experimental reproducibility, we set the Python pseudorandom seed at node startup (`random.seed(42)`), which makes the anomaly schedule and durations repeatable.

Anomaly activation (random duration, internal state updates, label publication, and logging) is detailed in Algorithm 5; the stop-and-go anomaly is formalized in Algorithm 6; and the CSV logging procedure is described in Algorithm 7, all in Appendix A.

Overall, controlled activation, reproducible simulation, and systematic logging support event traceability and provide a practical pipeline for generating labeled datasets used to train and evaluate the anomaly-detection models studied in this thesis.

4.7 Preprocessing: Dataset Cleaning and Preparation

At acquisition time, the dataset comprised 15,798 samples recorded at 10 Hz (1,579.8 s, ≈ 26.3 min). To train the autoencoder exclusively on nominal behavior and to mitigate startup transients, we discarded an initial warm-up window of 748 samples (74.8 s; $\approx 4.7\%$), yielding 15,050 nominal samples (≈ 25.1 min). Data integrity was then verified (absence of NaN/Inf values in the selected columns), and the sequence was ordered by `Timestamp`.

This cropping stabilizes the entropy metrics and improves the representativeness of the reference statistics used for thresholding (e.g., μ and σ). Moreover, it reduces the risk of bias by excluding non-representative transient segments that could otherwise contaminate the nominal subset.

4.8 Descriptive Stability of Entropy Metrics

To characterize the temporal regularity of entropy signals (Shannon, transition, and KDE), we evaluated two complementary measures: the mean absolute change, capturing

local smoothness via sample-to-sample absolute differences, and the standard deviation, quantifying global dispersion around the mean:

$$\text{Mean variability} = \frac{1}{N-1} \sum_{i=2}^N |E_i - E_{i-1}|, \quad \sigma_E = \sqrt{\frac{1}{N} \sum_{i=1}^N (E_i - \bar{E})^2},$$

where E_i denotes the entropy value at time i .

These measures were computed on the complete post-warm-up dataset (no subsampling), so the reported variability reflects the entire nominal sequence.

Table 4 summarizes the results. In terms of local smoothness, transition entropy shows the smallest mean change between samples; at the global level, Shannon entropy exhibits the lowest standard deviation. KDE is more sensitive to temporal fluctuations (higher variability and dispersion), suggesting that, although less stable, it may reflect subtle regime transitions.

Table 4: Stability measures for the entropy metrics.

Entropy metric	Mean variability	Standard deviation
Shannon	0.000714	1.77677×10^{-1}
Transition	0.000671	1.86974×10^{-1}
KDE	0.021522	1.04850

Figure 3 illustrates the evolution of the absolute sample-to-sample change. Flat segments indicate greater regularity; pronounced oscillations indicate episodes of local instability.

4.8.1 Stationarity Verification (ADF)

Before modeling, we verified stationarity using the Augmented Dickey–Fuller (ADF) test, which tests for a unit root. The estimated model was:

$$\Delta y_t = \alpha + \beta t + \gamma y_{t-1} + \sum_{i=1}^p \delta_i \Delta y_{t-i} + \varepsilon_t, \quad (7)$$

where $\Delta y_t = y_t - y_{t-1}$, p is the number of lags to control residual autocorrelation, and (α, β) allow for an intercept and deterministic trend. The null hypothesis is non-stationarity ($\gamma = 0$) versus the stationary alternative ($\gamma < 0$). Lag selection used an information criterion (autolag–AIC), and decisions relied on tabulated critical values and p -values [20, 30, 23].

Table 5 presents results for Shannon, transition, and KDE entropies. In all three cases, the ADF statistic was more negative than the 1% and 5% critical values, with p -values far below 0.05. Shannon and transition show stronger evidence (more negative statistics and

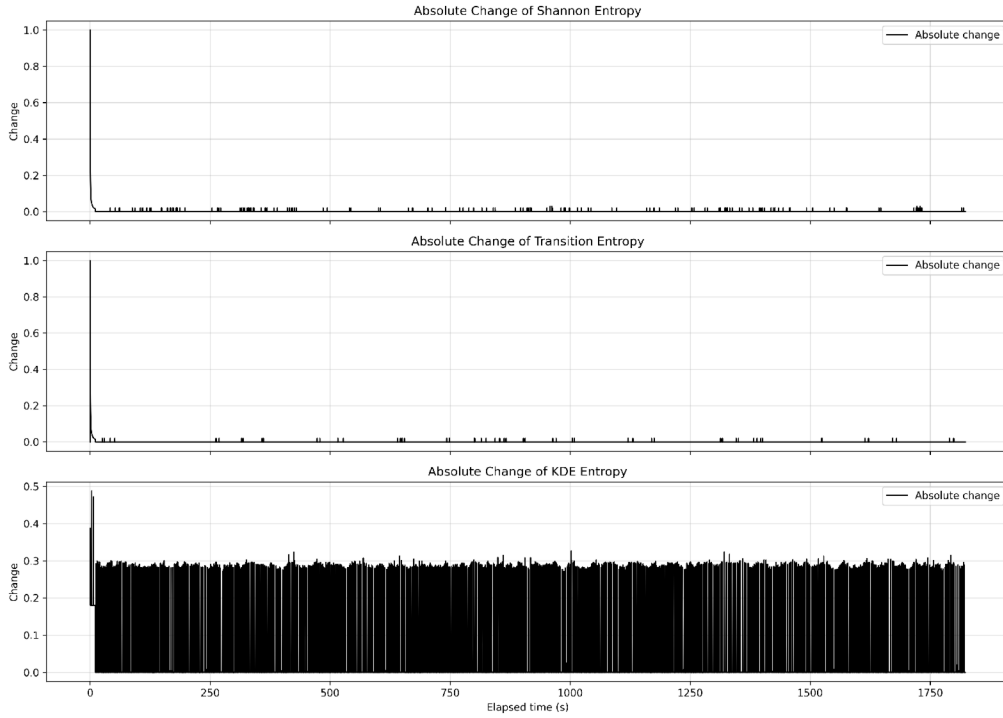


Figure 3: Absolute change between successive samples for each entropy (Shannon, transition, and KDE).

p -values $< 10^{-16}$) than KDE, which is nevertheless classified as stationary.

Table 5: ADF test results for entropies computed from robot behavior.

Entropy	ADF stat.	p -value	1%	5%
Shannon	-9.8706	4.00×10^{-17}	-3.4309	-2.8618
Transition	-10.2661	4.11×10^{-18}	-3.4309	-2.8618
KDE	-8.1937	7.55×10^{-13}	-3.4309	-2.8618

In summary, the three series behave as stationary over the analyzed segment. This supports their use as stable inputs for training and monitoring, and suggests a complementary role: Shannon and transition as robust references for calibration, and KDE as a more sensitive metric for subtle regime shifts.

4.8.2 Temporal Diagnosis of Entropy Metrics

As a post-preprocessing check (nominality filtering and burn-in trimming), we inspected the temporal stability of the entropy metrics. Figure 4 shows, for Shannon, transition, and KDE entropies, the raw series, its moving average, and the $\pm 1\sigma$ bands around that average. After the burn-in, all three metrics oscillate around stable levels; remaining micro-drops and spikes are associated with motion transitions or occasional irregularities in message cadence. This visual diagnosis supports using reference statistics computed on nominal segments and excluding the initial portion during training.

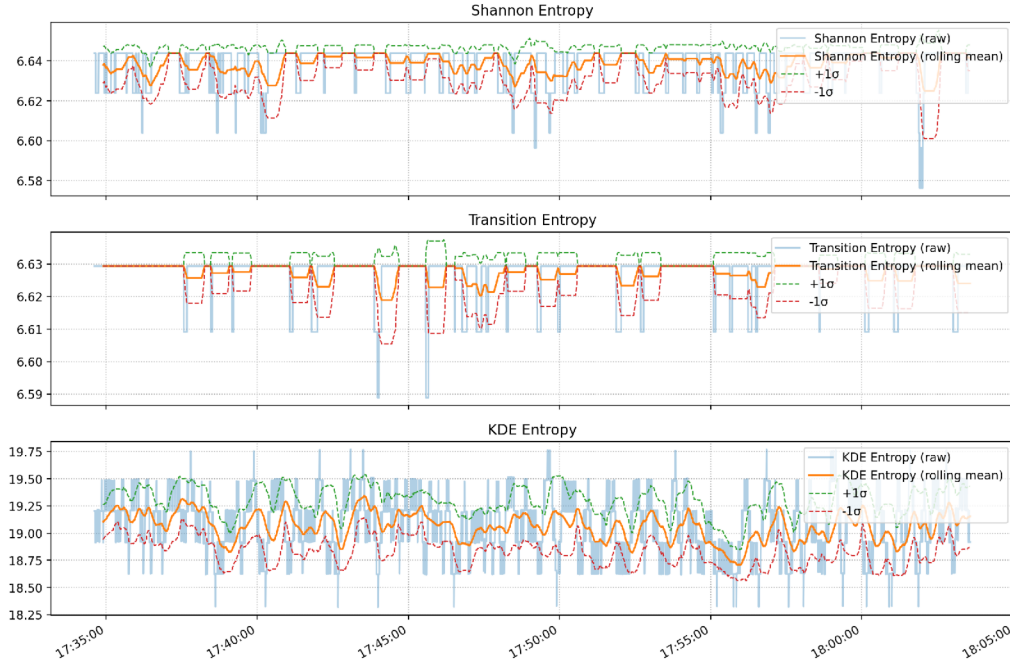


Figure 4: Temporal evolution of entropy metrics (Shannon, transition, and KDE) on nominal data: raw signal, moving average, and $\pm 1\sigma$ bands. Useful to verify stability after burn-in and to guide the calibration of reference statistics.

This procedure aligns with statistical process control practices (Shewhart-type $\mu \pm k\sigma$ bands and EWMA smoothing [56]) and with information-theoretic foundations for entropy usage [74, 16]. The KDE metric follows recent nonparametric entropic approaches [51]. Likewise, excluding a burn-in period prior to estimating statistics/thresholds follows the standard recommendation to remove initial transients until a steady regime is reached [60, 29].

4.9 Association Measures and Feature Selection

We adopt Spearman’s rank correlation (ρ) as the main measure of association between variables. Since the analysis resamples series to 20 Hz, organizes them in sliding windows, and the data exhibit pronounced transients, heavy tails, and potentially monotonic but not strictly linear relationships (e.g., between LiDAR minimum distance and linear speed), a rank-based measure is preferable. Spearman evaluates monotonic association from the relative ordering of observations and is therefore more robust than Pearson correlation to non-normality and moderate outliers.¹

As a robustness check we include Kendall’s τ_b , which explicitly corrects for ties and provides a complementary contrast when discretized variables or derived, quantized metrics are present. We adopt a conservative criterion: relevant association blocks identified

¹Rank-based correlations (Spearman, Kendall) are less sensitive to deviations from normality and to local ties induced by quantization or thresholds.

with Spearman should be preserved—at least qualitatively—under Kendall; when substantive discrepancies arise, we report and discuss their implications for the stability of conclusions.

Pearson correlation is reserved for situations where linearity is plausible and useful (e.g., stationary segments within a single motion state) or for linear diagnostics such as the Variance Inflation Factor (VIF) to assess multicollinearity, avoiding its use as justification for relationships that are in fact nonlinear but monotonic. In feature construction we also avoid introducing a metric and its thresholded derivative simultaneously, to reduce collinearity and minimize information leakage into the model.

Figure 5 presents the Spearman (ρ) correlation heatmap, accompanied in the text by a qualitative summary of the most associated pairs and the blocks emerging from the matrix. Figure 6 shows an analogous heatmap computed with Kendall’s τ_b , whose qualitative agreement with the main result acts as a robustness check. When appropriate, point observations with Pearson are reported in stratified analyses or in the context of linear diagnostics.

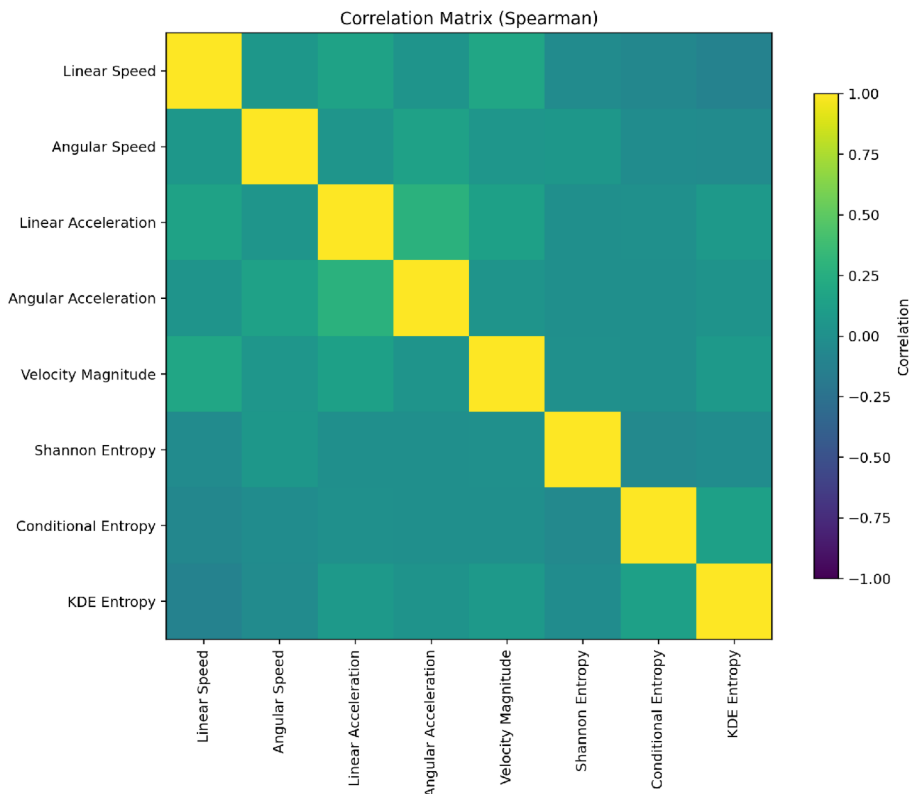


Figure 5: Correlation matrix (Spearman, ρ). Color coding in $[-1, 1]$ highlights association blocks, inverse relations, and redundancies across variables.

Although the Spearman and Kendall heatmaps appear visually similar, this is expected because both are rank-based estimators of monotonic association and typically differ only slightly in magnitude. To make these small discrepancies explicit, we also report estimator-difference maps: the signed difference ($\rho - \tau_b$) and the absolute difference

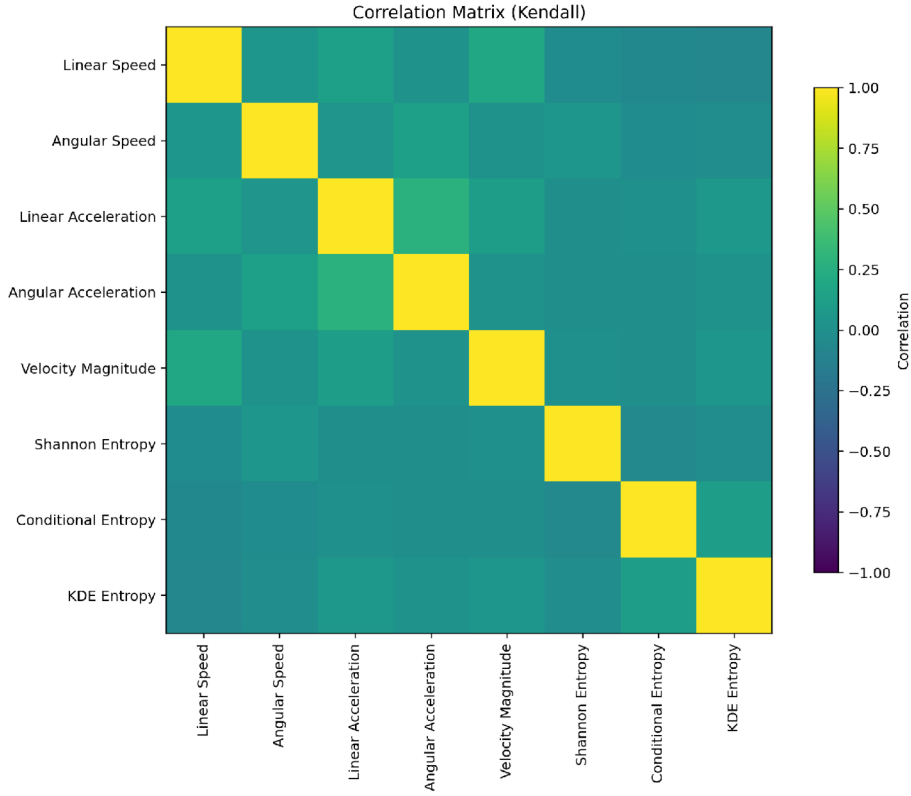


Figure 6: Robustness check with Kendall (τ_b). Qualitative preservation of the main blocks detected with Spearman is observed.

$|\rho - \tau_b|$ (Figure 7). In our data, these differences are small and localized, which supports the robustness of the qualitative block structure used for feature selection.

Quantitatively, the average absolute difference across unique variable pairs is ≈ 0.010 , with a maximum of ≈ 0.035 .

4.9.1 Correlation- and Relevance-Based Selection (8 variables)

Based on the Spearman (ρ) analysis, we adopt a parsimonious criterion to reduce multicollinearity and avoid leakage, retaining only variables that contribute complementary signal to the AE-LSTM pipeline. The correlation matrix, ordered via hierarchical clustering, reveals two main informative blocks: a kinematic block describing motion state and local dynamics, and an entropy block sensitive to the temporal regularity of message flow and operating-regime shifts. Accordingly, the final selection is: *linear speed*, *angular speed*, *velocity magnitude*, *linear acceleration*, *angular acceleration*, *Shannon entropy*, *transition entropy*, and *KDE entropy*.

We excluded derived threshold columns (e.g., *Entropy Threshold Lower/Upper* and *KDE Threshold Lower/Upper*) because they are deterministic transformations of the base metrics and thus strongly collinear. We also excluded jerk; although conceptually tied to acceleration (time derivative), in our records its association is moderate ($\rho \approx 0.64$) and its high sensitivity to noise can amplify false positives without additional smoothing.

Estimator-dependent differences between Spearman and Kendall

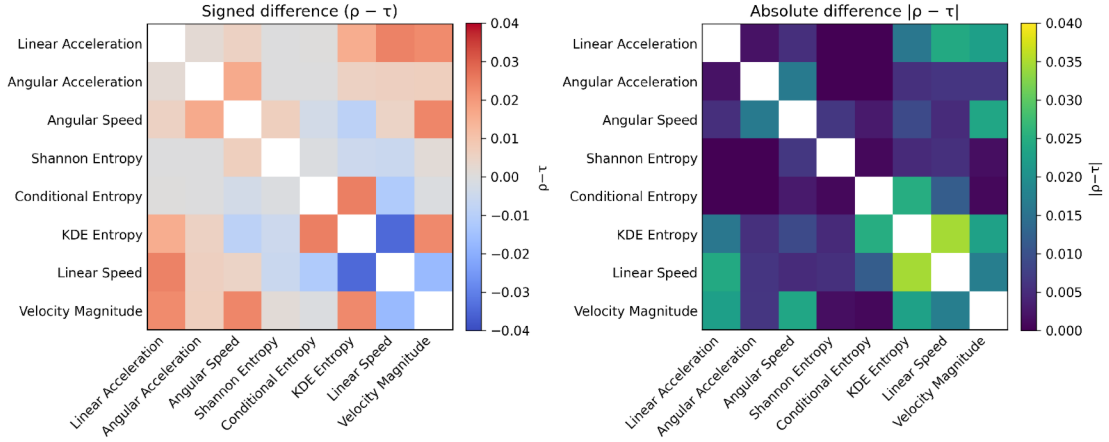


Figure 7: Estimator-dependent differences between Spearman (ρ) and Kendall (τ_b). Left: signed map ($\rho - \tau_b$). Right: magnitude map $|\rho - \tau_b|$. Differences are small overall and concentrated in a few pairs, hence the qualitative dependence structure is preserved across estimators.

The retained set preserves essential kinematic signal and temporal-irregularity information without strong redundancies, favoring a stable latent representation and more robust threshold calibration during detection.

4.10 Models and Training

To detect anomalous behavior in the robot-generated data, we adopt a sequential approach with two models trained exclusively on nominal data: an autoencoder (AE) and an LSTM predictor. We train an overcomplete dense autoencoder whose encoder maps the input vector $x_t \in \mathbb{R}^8$ to a higher-dimensional latent space $z_t \in \mathbb{R}^{10}$, and whose decoder reconstructs $\hat{x}_t \in \mathbb{R}^8$. This design prioritizes embedding expressiveness under nominal regime; to discourage the identity mapping and favor generalization we applied regularization (weight decay, dropout, early stopping) together with prior standardization of input variables. The AE thus acts as a stable representation extractor, attenuating noise and spurious correlations.

On the latent sequence $\{z_t\}$ produced by the encoder, we train an LSTM that, from fixed-length windows T , forecasts the next embedding: given $(z_{t-T+1}, \dots, z_t)$, it estimates $\hat{z}_{t+1}$ and learns the typical temporal dynamics under nominal conditions. At inference time, the pipeline applies the encoder and LSTM to new traces (with possible anomalies) and computes the latent prediction error

$$e_{t+1} = \|z_{t+1} - \hat{z}_{t+1}\|_2.$$

The decision is binarized with a statistically calibrated threshold on nominal validation,

$$\text{threshold} = \mu + 3\sigma,$$

where μ and σ are the mean and standard deviation of $\{e_{t+1}\}$ on that set. This threshold is stored and reused to evaluate new traces without requiring additional normal-data recalibration. Each instance is labeled as normal or anomalous depending on whether e_{t+1} exceeds the threshold, producing an `is_anomaly` column in the output file for subsequent analysis.

4.10.1 Autoencoder

The autoencoder was trained on the eight variables capturing system kinematics and temporal regularity: linear speed, angular speed, linear acceleration, angular acceleration, velocity magnitude, Shannon entropy, transition entropy, and KDE entropy. A `StandardScaler` was fit exclusively on the training segment, and we used an 80/20 temporal split (no leakage), reserving the last 20% for validation.

Let $x_t \in \mathbb{R}^8$ be the normalized feature vector at time t . We employ an *overcomplete* autoencoder that performs a nonlinear feature expansion: the encoder implements $f_{\text{enc}} : \mathbb{R}^8 \rightarrow \mathbb{R}^{10}$ and produces the latent embedding $z_t \in \mathbb{R}^{10}$, while the decoder implements $f_{\text{dec}} : \mathbb{R}^{10} \rightarrow \mathbb{R}^8$ to reconstruct $\hat{x}_t = f_{\text{dec}}(z_t)$. Rather than enforcing strict dimensional compression, the objective is to learn a structured representation that is convenient for downstream temporal modeling (LSTM) while preserving reconstructability. To discourage trivial identity mappings, we use dropout ($p = 0.10$) and rely on validation-guided training (early stopping), weight decay, and validation-based calibration of reconstruction-error thresholds.

Figure 8 summarizes the overcomplete autoencoder architecture used in our pipeline. The training objective minimizes the mean squared error (MSE) between each input vector and its reconstruction, defined as:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{N} \sum_{t=1}^N \|x_t - \hat{x}_t\|_2^2, \quad \hat{x}_t = f_{\text{dec}}(f_{\text{enc}}(x_t)), \quad (8)$$

where N is the number of training samples, x_t is the normalized input at time step t , and $\hat{x}_t$ is its reconstruction.

We used Adam with learning rate 10^{-3} and weight decay 10^{-5} , batch size 256, and up to 150 epochs. Learning-rate scheduling was performed with `ReduceLROnPlateau` (factor 0.5, patience 8 epochs) on the validation MSE, and early stopping with a patience of 20 epochs. In addition to MSE, the mean absolute error (MAE) was tracked per epoch. Training curves are shown in Figures 9 (MSE loss) and 10 (MAE). A reconstruction threshold for diagnostics was calibrated on the validation set from the empirical distribution

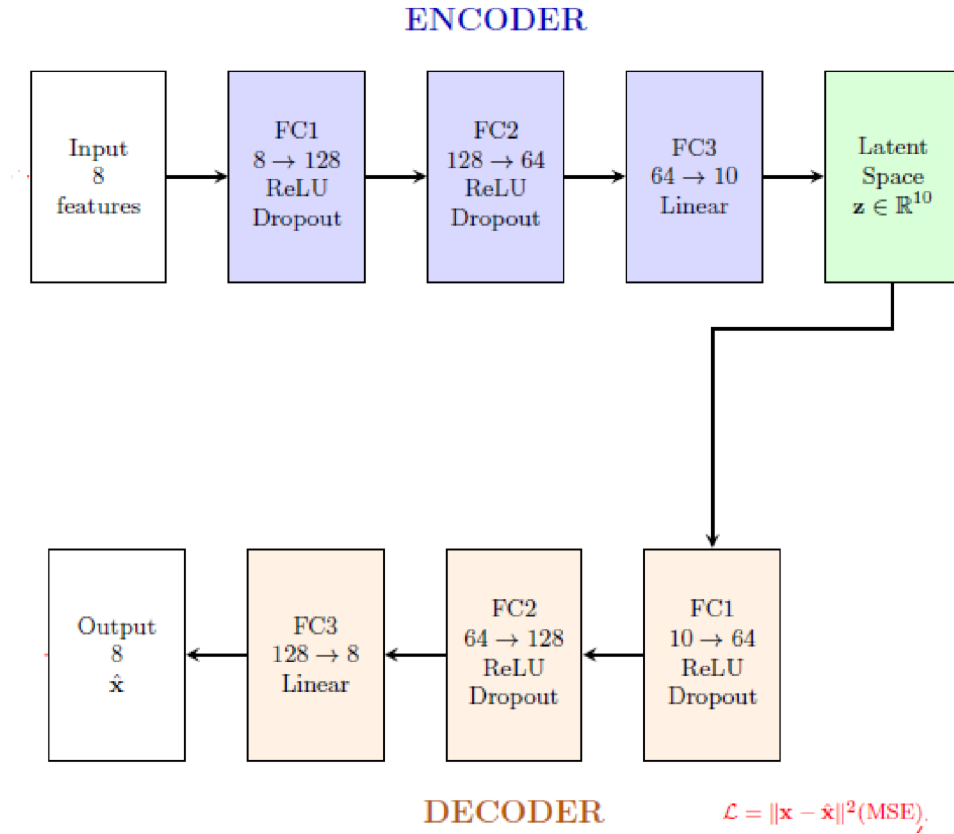


Figure 8: Overcomplete autoencoder used in the pipeline. The encoder maps $x_t \in \mathbb{R}^8$ to a latent embedding $z_t \in \mathbb{R}^{10}$ (MLP $8 \rightarrow 128 \rightarrow 64 \rightarrow 10$) and a symmetric decoder reconstructs $\hat{x}_t$ ($10 \rightarrow 64 \rightarrow 128 \rightarrow 8$). Dropout ($p = 0.10$) is applied in hidden layers during training, and the model is optimized with MSE reconstruction loss.

of MSE, storing μ , σ , and $\tau = \mu + 3\sigma$ as operational references.

Post-training artifacts include the feature scaler, the full autoencoder and its encoder (including a serialized deployment version). Outputs also include loss curves and an epoch-wise metrics table. Finally, latent embeddings $\{z_t\}$ for the training, validation, and full sets were exported in temporal order; these serve as input to the LSTM predictor.

On the overcomplete latent ($8 \rightarrow 10$): rather than strict compression, the encoder performs a lifting to a higher-dimensional space that acts as learned nonlinear feature engineering, $z_t = \phi(x_t)$. This dimensionality increase can render dynamics more regular (closer to linear) and separate factors of variation (kinematics vs. temporal irregularity), facilitating the prediction of z_{t+1} from z_t . The risk of trivial solutions is mitigated with dropout, weight decay, early stopping, and temporal validation. Empirically, the suitability of an overcomplete latent is verified by comparing (i) reconstruction loss on validation and (ii) LSTM predictor performance in latent space (lower error and improved robustness to transients).

The latent lifting ($\mathbb{R}^8 \rightarrow \mathbb{R}^{10}$) is analogous to mapping into a feature space where dynamics become more regular and approximately linear, easing latent-space forecasting.

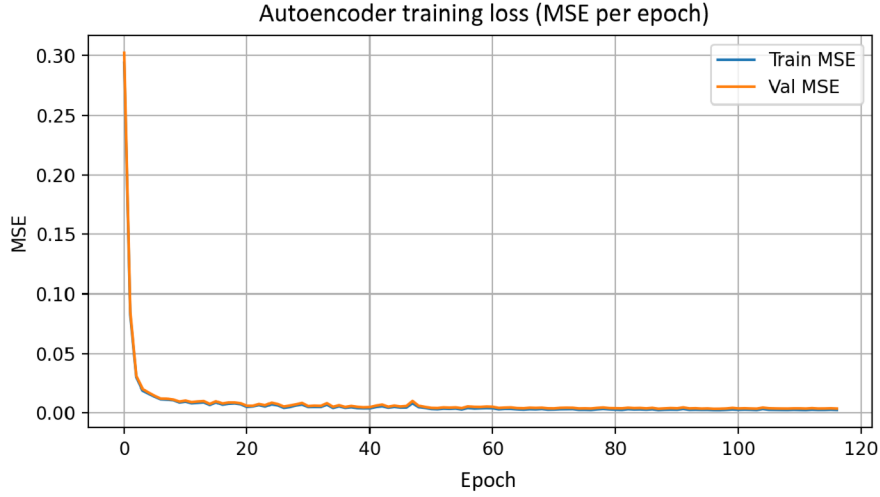


Figure 9: Autoencoder loss evolution during training (MSE per epoch).

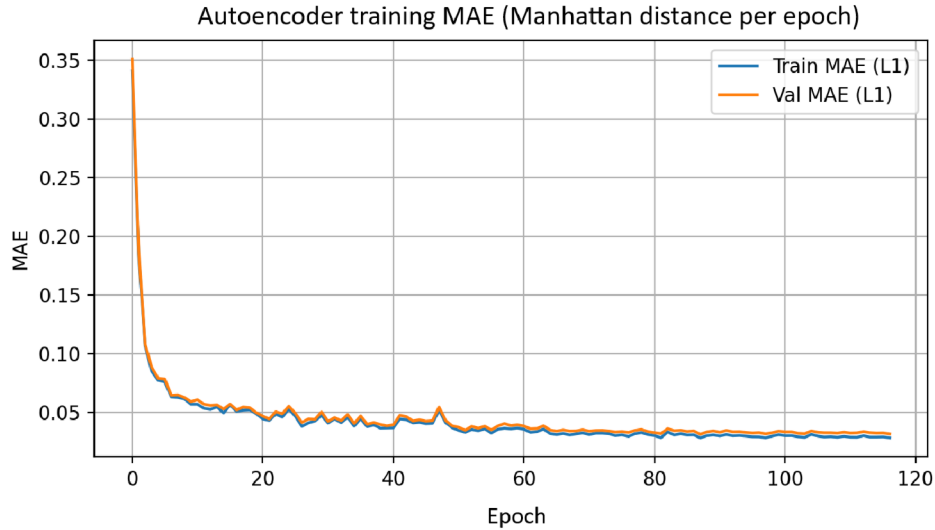


Figure 10: Manhattan distance (MAE) per epoch during autoencoder training.

Representation-learning literature documents overcomplete representations and the need for constraints (denoising, sparsity, regularization) to avoid identity solutions [9, 79, 57]. The idea is akin to kernels mapping to higher-dimensional spaces [71, 63] and to Koopman/EDMD approaches in dynamical systems that lift nonlinear dynamics to an approximately linear regime [81]. For clarity, we refer to $z_t \in \mathbb{R}^{10}$ (encoder output) as latent embeddings. In our pipeline, the LSTM operates exclusively on these embeddings; the decoder is only used for reconstruction $\hat{x}_t \in \mathbb{R}^8$ and reconstruction-error estimation, not for temporal prediction. The end-to-end autoencoder training/export procedure is summarized in Algorithm 8 (Appendix A).

4.10.2 LSTM

The LSTM model was trained on sequences of latent embeddings produced by the autoencoder. The embedding time series is split into training and validation subsets while preserving temporal order; in our workflow we use an 80–20 split (first 80% of the sequence for training and the remaining 20% for validation). When precomputed embeddings (`embeddings_{train, val}.npy`) are available, the script loads these files directly; otherwise, it generates embeddings from normalized data via the autoencoder and constructs the same 80–20 temporal partition. A `StandardScaler` is fit exclusively on the training embeddings and then reused to transform the validation set, avoiding leakage.

Supervised instances use length $T = 20$ with the scheme

$$\mathbf{X} = \mathbf{Z}_{t-T+1:t} = (z_{t-T+1}, \dots, z_t) \in \mathbb{R}^{T \times 10} \longrightarrow \mathbf{Y} = z_{t+1} \in \mathbb{R}^{10},$$

i.e., from each window of 20 consecutive embeddings we predict the next embedding.

The architecture is a single-layer LSTM with $H = 32$ hidden units and a final linear projection operating in residual form:

$$\hat{z}_{t+1} = z_t + \Delta_{\theta}(\mathbf{Z}_{t-T+1:t}),$$

where Δ_{θ} is the correction estimated from the last hidden state.

Figure 11 summarizes the residual LSTM predictor; note that embeddings are standardized using a `StandardScaler` fitted on the training set and reused for validation.

Training optimizes the Smooth L1 (Huber) loss with Adam ($\eta = 5 \times 10^{-4}$, weight decay 10^{-4}), batch size = 128, up to 100 epochs. Hyperparameters were initialized using commonly adopted defaults and subsequently tuned empirically using the training/validation curves, prioritizing stable convergence and avoiding overfitting. We included `ReduceLROnPlateau` (factor 0.5, patience 8), early stopping (patience 10), and gradient clipping (norm 1.0). For each epoch we compute MSE and MAE on the training loader (“Train MSE/MAE”) and on the validation loader (“Val MSE/MAE”), where each metric is the average per-sample error over all sequences in the corresponding subset.

Artifacts include LSTM predictor weights (state dict and full module) and a serialized inference version, the latent-space scaler, operational thresholds calibrated on validation (error-distribution statistics, e.g., μ , σ , $\mu+3\sigma$, $p99$, $p99.7$), training curves and epoch-wise metrics, and configuration metadata (latent dimension, T , H , learning rate, batch size, patience, weight decay, etc.). Figure 12 shows the MSE loss evolution.

Behaviorally, the predictor approximates the nominal one-step evolution law in latent space,

$$\hat{z}_{t+1} = F_{\theta}(\mathbf{Z}_{t-T+1:t}) \approx z_{t+1},$$

capturing regular temporal patterns (trends and cross-component dependencies within the

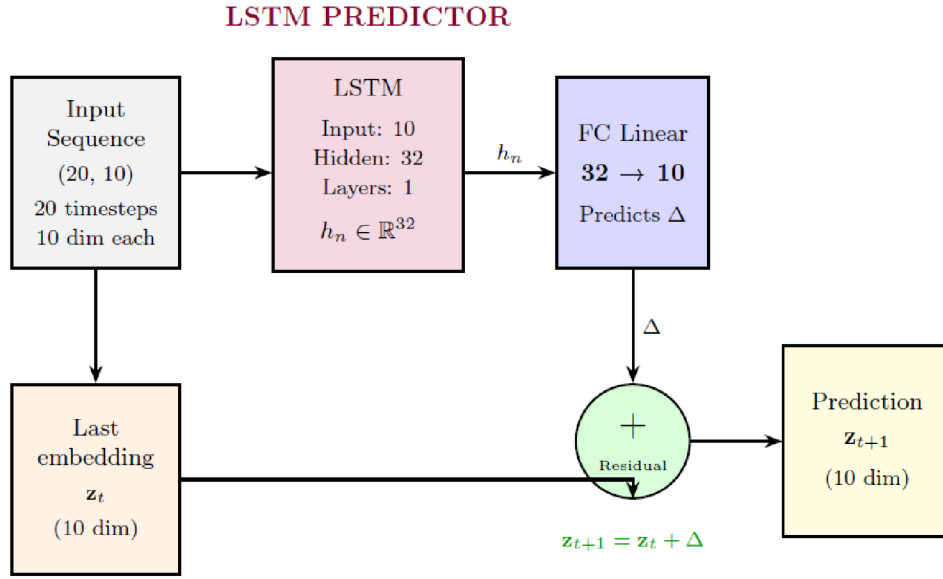


Figure 11: Residual LSTM predictor used to forecast the next latent embedding. A window of $T = 20$ *standardized* embeddings ($\mathbf{Z}_t \in \mathbb{R}^{20 \times 10}$, normalized with a `StandardScaler` fitted on the training set) is processed by a one-layer LSTM ($H = 32$). The last hidden state is mapped by a fully connected layer ($32 \rightarrow 10$) to estimate a residual increment Δ_t , and the prediction is formed as $\hat{\mathbf{z}}_{t+1} = \mathbf{z}_t + \Delta_t$, where $\mathbf{z}_t$ is the last embedding in the input window.

embedding) by training solely on nominal data.

The output $\hat{\mathbf{z}}_{t+1} \in \mathbb{R}^{10}$ enables an anomaly score defined as the latent residual,

$$e_{t+1} = z_{t+1} - \hat{z}_{t+1}, \quad S_{t+1} = \|e_{t+1}\|_2^2,$$

and triggers an alert if $S_{t+1} > \tau$, with τ calibrated on validation (e.g., $\mu + 3\sigma$). Optionally, $\hat{\mathbf{z}}_{t+1}$ can be projected through the decoder to obtain $\hat{\mathbf{x}}_{t+1}$ and aid diagnostics of variables departing from the nominal regime.

Two independent scalings are used in the pipeline: a feature scaler applied to $x_t \in \mathbb{R}^8$ before the encoder (stabilizes AE loss) and a latent scaler applied to $z_t \in \mathbb{R}^{10}$ before the LSTM (equalizes latent variances and improves numerical stability). The decoder operates on z_t without the latent-scaler normalization only for reconstruction; the score S_{t+1} is computed consistently (either in normalized or de-normalized latent space). The full LSTM training/export procedure is detailed in Algorithm 9 (Appendix A).

4.10.3 Robust standardization and composite anomaly score

After characterizing statistical and temporal regularity via sliding-window entropy metrics, it is natural to incorporate a model that simultaneously captures (i) the data's latent structure and (ii) its sequential dynamics.

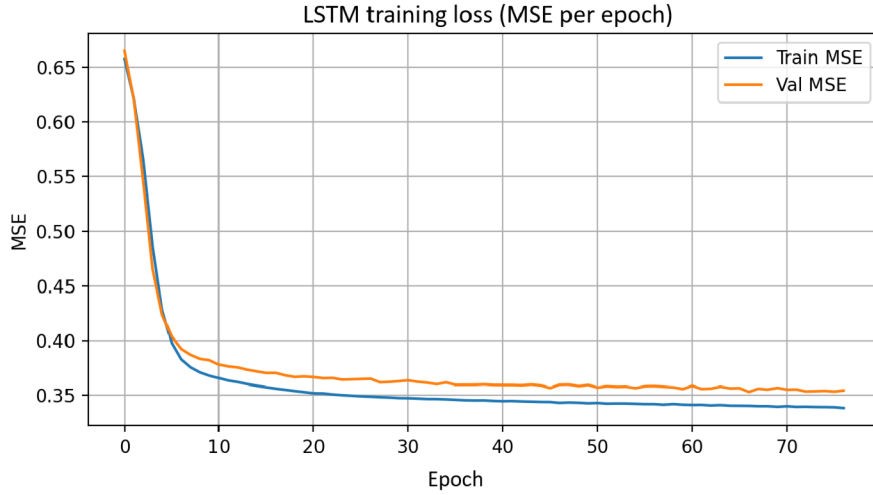


Figure 12: Loss evolution (MSE) per epoch during LSTM predictor training.

To that end, we adopt a hybrid architecture comprising an Autoencoder (AE) and an LSTM network. The AE learns, unsupervised, a nominal latent representation of telemetry variables and entropy metrics; its reconstruction error acts as a direct indicator of deviation from the normal regime [31, 28]. To enhance representational capacity without sacrificing generalization, we use an overcomplete regularized AE (L2 penalty, dropout, early stopping), a strategy supported by the robust autoencoder literature (denoising, sparsity, contractive) [79, 55, 65]. This regularization discourages trivial identity mappings and helps the network capture stable regularities rather than fitting idiosyncratic patterns in the training set.

The latent vector produced by the AE feeds an LSTM trained for one-step prediction, so that the prediction error reflects dynamic deviations (e.g., regime change, sustained jitter, stop-go patterns) [32]. The convenience of combining reconstruction (AE) and prediction (LSTM) is well documented in time-series anomaly detection across industrial and robotic domains, where AE/LSTM variants—including VAE-LSTM—have shown high sensitivity with unlabeled data [70, 48, 33, 59, 39, 41].

Let e_t^{AE} denote the AE reconstruction error and e_t^{LSTM} the one-step prediction error in the latent space. For each error source, we denote by $e^{(\cdot)} = \{e_t^{(\cdot)}\}_{t=1}^T$ the corresponding error sequence, with $(\cdot) \in \{AE, LSTM\}$, and by $\text{med}(\cdot)$ its sample median. We then obtain robust z -scores by standardizing each sequence using the median and the median absolute deviation (MAD),

$$\text{MAD}(e^{(\cdot)}) = \text{med}_t \left(|e_t^{(\cdot)} - \text{med}(e^{(\cdot)})| \right),$$

so that

$$z_t^{(\cdot)} = \frac{e_t^{(\cdot)} - \text{med}(e^{(\cdot)})}{\text{MAD}(e^{(\cdot)}) + \varepsilon},$$

with $\varepsilon > 0$ to avoid division by zero. The composite score is defined as a weighted sum,

$$S_t = \alpha z_t^{AE} + \beta z_t^{LSTM}, \quad \alpha, \beta \geq 0.$$

In this formulation, $\alpha + \beta = 1$ is not required, since any global rescaling of S_t can be absorbed into the offline threshold calibration. The score S_t is then compared to a threshold t_h under a persistence rule K : an anomaly is declared only if $S_t > t_h$ in at least K of the last W samples. This rule damps spurious triggers and stabilizes online decisions, aligning with change-detection practice and robust operational triggering [6, 33]. The hyperparameters (α, β, t_h, K) are calibrated offline with nominal data (and, when available, synthetic injection scenarios), maximizing recall subject to an explicit per-window FPR bound.

In summary, the AE provides a regularized, stable latent representation of the nominal regime (not implying strict compression when overcomplete, but a lifting that attenuates noise and spurious correlations while flagging deviations via reconstruction); the LSTM captures fine temporal coherences (flagging deviations via prediction); and the assembly with robust normalization and persistence K yields a decision score suitable for real-time operation on ROS 2/DDS, complementing entropy metrics with a sequential mechanism featuring low latency and explicit false-alarm control.

Figure 13 summarizes the end-to-end ADS flow on ROS 2/DDS and situates the proposed AE–LSTM Combination module within the architecture.

4.10.4 AE–LSTM Flow Summary

Figure 14 summarizes the complete pipeline. First, raw features $x_t \in \mathbb{R}^8$ are normalized using a `StandardScaler`: for each dimension, the training-set mean is subtracted and the result is divided by the corresponding standard deviation, and the same affine transformation is then applied to validation and online data. The AE encoder then projects these standardized features into an embedding $z_t \in \mathbb{R}^{10}$. For temporal prediction, z_t is re-scaled with a latent scaler and organized into windows of length $T = 20$ that feed the LSTM. The predictor operates in residual form,

$$\hat{\tilde{z}}_{t+1} = \tilde{z}_t + \Delta_{\theta}(\tilde{\mathbf{Z}}_{t-T+1:t}),$$

and its output enables an anomaly score defined as a latent-space residual. The decoder branch (dashed) is optional in operation and used for reconstruction and diagnostics. This flow integrates training, deployment, and threshold calibration $(\mu + 3\sigma)$ within a stable latent representation, contributing to low detection latency and sensitivity to subtle deviations, while emphasizing the importance of temporal alignment and drift monitoring.

ROS2 / DDS Anomaly Detection and Alerting Pipeline

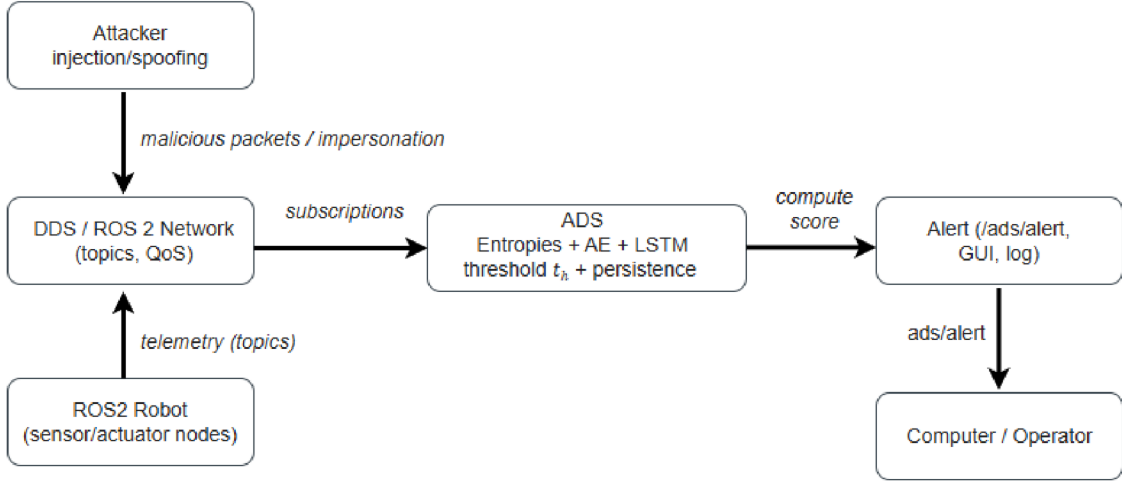


Figure 13: System-level data flow of the Anomaly Detection System (ADS) on ROS 2/DDS. Telemetry published by the ROS 2 robot is transported over the DDS network and subscribed by the ADS, which computes a detection score by combining entropy-based descriptors with an Autoencoder (AE) and an LSTM predictor. A binary anomaly decision is obtained by comparing the score with a threshold t_h under a persistence rule K . When an anomaly is detected, the ADS publishes an alert (`/ads/alert`) to the operator’s GUI and logs. The main attack surface, where an adversary can inject or spoof DDS traffic, is indicated on the left.

4.11 Offline Experiments: Objectives, Thresholds, and Metrics

4.11.1 Rationale for Offline Calibration

Anomaly detection on time series here relies on two complementary signals: the reconstruction error of an autoencoder trained on nominal data and the prediction error of a recurrent model capturing temporal dynamics. The first reflects deviations in observation space; the second, inconsistencies in latent evolution. Both approaches have shown effectiveness in multivariate and telemetry contexts [49, 33, 12].

To stabilize error scales under heavy tails and outliers, we apply robust standardization based on the median and the median absolute deviation (MAD) [68]. For each error sequence e_t (either AE reconstruction or LSTM one-step prediction error), let

$$\text{MAD}(e) = \text{med}_t(|e_t - \text{med}(e)|), \quad (9)$$

and define the robustly standardized error as

$$\tilde{e}_t = \frac{e_t - \text{med}(e)}{\text{MAD}(e) + \varepsilon}, \quad \varepsilon > 0. \quad (10)$$

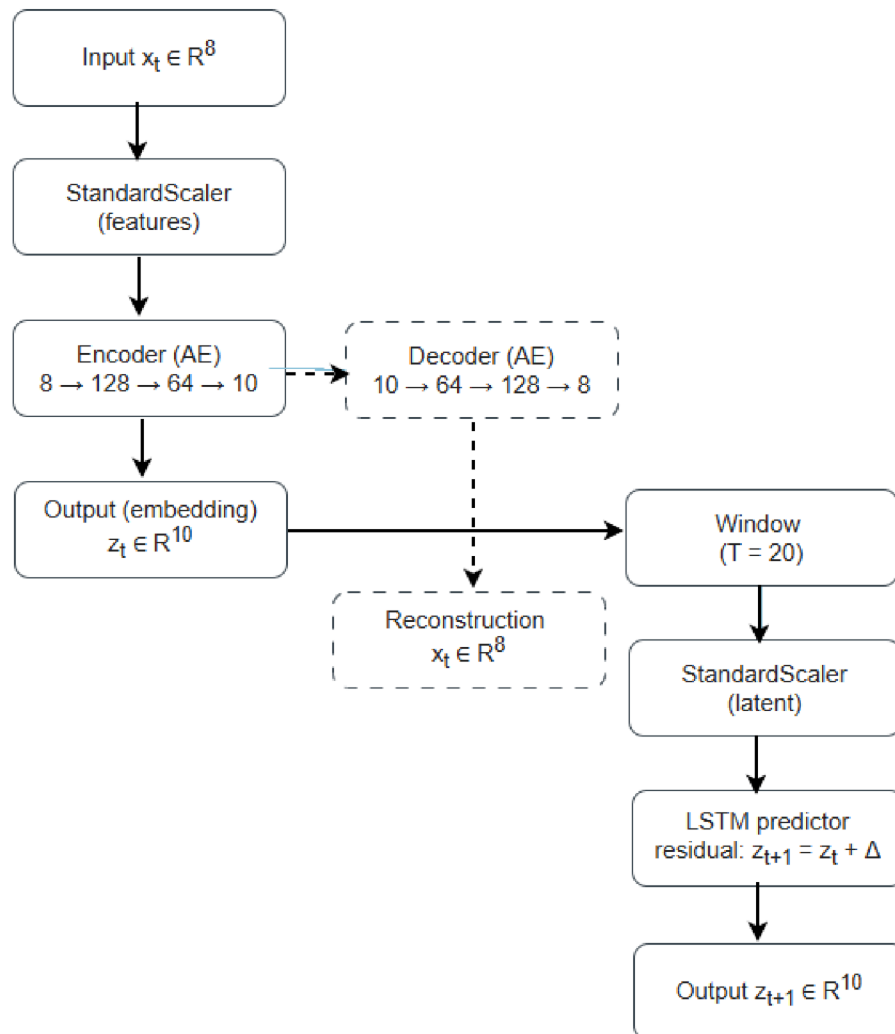


Figure 14: End-to-end AE-LSTM flow.

Denoting the resulting standardized series by $\tilde{e}_{\text{AE},t}$ and $\tilde{e}_{\text{LSTM},t}$, we combine them into the score

$$S_t = \alpha \tilde{e}_{\text{AE},t} + \beta \tilde{e}_{\text{LSTM},t}, \quad \alpha, \beta \geq 0. \quad (11)$$

In this formulation, $\alpha + \beta = 1$ is not required, since any global rescaling of S_t can be absorbed into the offline threshold calibration.

The decision is obtained by thresholding S_t and applying a persistence rule: K consecutive exceedances are required to declare an alarm, analogous to run rules in statistical process control to mitigate false alarms [54].

Threshold selection is posed as an operating-point problem under an ROC criterion, prioritizing high sensitivity with an explicit bound on the false positive rate, in line with classical ROC literature and Type I error control [24]. Given the temporal nature of anomalies, we complement window-level metrics with range-/event-level metrics [77] for a more faithful report.

4.11.2 Sequential Offline Calibration Procedure

Calibration proceeds in two decoupled stages:

Stage 1 (error collection). We run inference on the evaluation set without making decisions, solely to log, per temporal window, the latent prediction error (e_{LSTM}) and the autoencoder reconstruction error (e_{AE}). These values are stored along with timestamps and, when available, labels. Any operational parameters appearing in the script (α, β, K, t_h) do not intervene in computing e_{LSTM} or e_{AE} , so the output is a neutral input for calibration.

Stage 2 (operating-point search). From the error logs, we apply robust standardization (Eqs. (9)–(10)) and construct the score $S_t(\alpha, \beta)$ (Eq. (11)). For each pair (α, β) in a finite grid and for each candidate persistence value K , we sweep thresholds t_h defined by score percentiles. When labels exist, the threshold is chosen to maximize sensitivity under a pre-specified FPR bound; ties are broken by F_1 . If no threshold satisfies the bound, we select the one with the minimum FPR, with ties broken by sensitivity.

The process yields a result table summarizing, for each (α, β, K) , the selected threshold and associated metrics, together with the normalization statistics used. With the resulting quartet $(\alpha^*, \beta^*, K^*, t_h^*)$ we rerun the offline inference to produce the consolidated report.

4.11.3 Description of Offline Inference

This module implements inference and evaluation without online decision-making on a labeled dataset. Its goal is to generate, per temporal window, the necessary error signals and the combined score for the subsequent calibration stage. We assume the availability of autoencoder artifacts (feature scaler, serialized encoder, and for reconstruction error the full model or its state) and of the LSTM (latent-space scaler and serialized predictor),

as well as the temporal window size and latent dimension.

The flow is as follows. First, we load and clean the relevant variables from the evaluation set; features are standardized with the autoencoder’s `StandardScaler` and projected with the encoder to obtain the latent sequence $\{z_t\}$. Then embeddings are standardized with the LSTM scaler and organized into fixed-length windows to predict z_{t+1} . The prediction error is computed as the Euclidean norm between the true value and the LSTM estimate; in parallel, the autoencoder reconstruction error (instance-wise RMSE) is computed and temporally aligned with the windows. Both errors are robustly standardized (Eqs. (9)–(10)) and combined into S_t as in Eq. (11). From S_t , we perform a percentile threshold sweep to compute window-level metrics when labels are present. Finally, persistent outputs include the time series of errors and scores (with timestamps and, if available, labels), the threshold-sweep summary with metrics, supporting plots (histograms and time series), and configuration metadata. Operational pseudocode is provided in Algorithm 10 in Appendix A.

4.11.4 Calibration of Weights, Persistence, and Threshold

This module performs offline calibration of the detector’s operating hyperparameters using the error logs produced in the previous stage. As input, it uses the per-window error series and, when available, binary reference labels. Both errors are robustly standardized (Eqs. (9)–(10)) and combined into the score (Eq. (11)). For a finite grid of (α, β) pairs and persistence values K , we sweep thresholds defined by percentiles of S_t and emulate the persistence logic: an alarm is declared when K consecutive exceedances are observed. If labels exist, the threshold for each (α, β, K) is chosen to maximize sensitivity under a pre-specified FPR bound; ties are broken by F_1 . If no configuration satisfies the bound, we choose the threshold with minimum FPR and, in case of ties, the one with higher sensitivity. Operational pseudocode is given in Algorithm 12 in Appendix A.

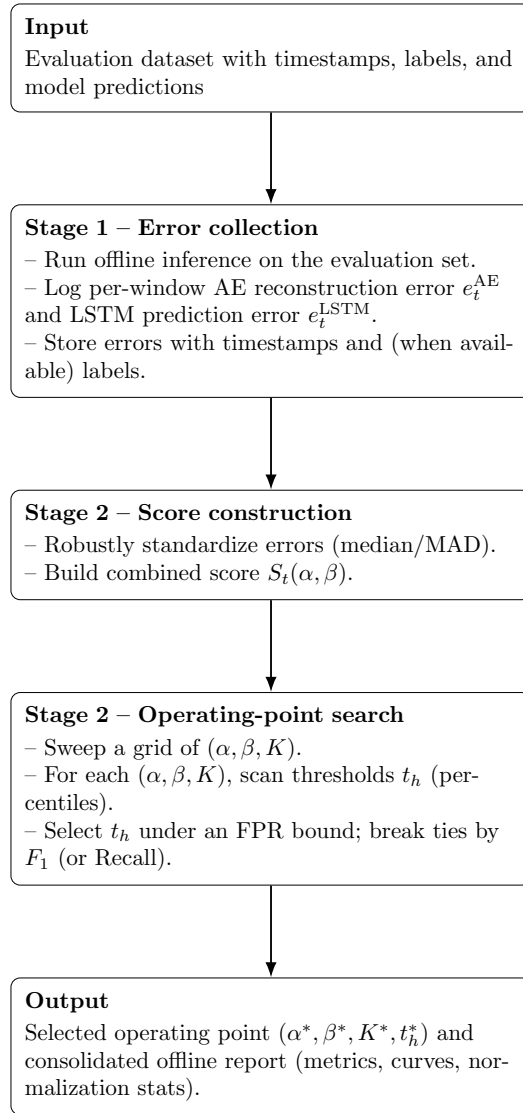


Figure 15: High-level overview of the two-stage offline calibration procedure: Stage 1 logs raw AE/LSTM errors on the evaluation set; Stage 2 applies robust standardization, combines them into the score $S_t(\alpha, \beta)$, and sweeps thresholds and persistence K under an FPR constraint.

5 EVALUATION

This chapter presents the objectives and the experimental setup of the offline evaluation in the simulated ROS 2-based mobile robot navigation scenario used in this work, including normal operation and injected anomalous episodes, the window-level and event-level results, and their discussion. We analyze temporal properties, stationarity, operating-point selection, and deployment considerations.

5.1 Objectives and Experimental Setup

The purpose of this evaluation is to quantify three measurable aspects: (i) the temporal stability of the entropy metrics under nominal conditions (robustness of the entropy inputs); (ii) the detection sensitivity of the Autoencoder–LSTM scheme under an explicit operating constraint (offline-calibrated to satisfy a bounded per-window FPR); and (iii) event-level responsiveness, quantified by detection latency and event hit rate.

The experimental setup follows the Methodology (§4.8–§4.11.4): data were simulated in ROS 2 Humble with Fast DDS; acquired at 10 Hz and resampled to 20 Hz for analysis; and organized into temporal windows of length $T=20$. Binary decisions are produced by robust median/MAD normalization [68] of the AE and LSTM error signals, followed by score combination and thresholding with a persistence rule requiring K consecutive exceedances. The operating point was calibrated *offline* to maximize *recall* subject to an explicit bound on FPR (ties resolved by F_1), and was kept fixed during the test runs. Reporting includes window-level metrics (precision, *recall*, F_1 , FPR), in line with ROC practices [24], and event-level metrics (detection latency, coverage, and *event hit rate*) consistent with the temporal nature of anomalies [77].

5.1.1 Evaluation Metrics

Evaluation is conducted on the decision series generated by the combined score and the persistence rule, applied to the test data in temporal order. Let $y_t \in \{0, 1\}$ be the ground-truth window label and $\hat{y}_t \in \{0, 1\}$ the detector’s decision after thresholding S_t and applying persistence K .

From the confusion matrix (true positives, false positives, true negatives, and false negatives) we report standard window-level metrics: precision $P = TP/(TP + FP)$, recall $R = TP/(TP + FN)$, and $F_1 = 2PR/(P + R)$. To explicitly control spurious triggers under nominal regime, we report the false positive rate $FPR = FP/(FP + TN)$ (equivalent to $1 - \text{specificity}$). As context we report prevalence $\pi = (TP + FN)/N$, where N is the total number of windows. All metrics are computed *after* applying persistence K , since this mechanism is part of the effective decision logic and reduces isolated activations at the potential cost of delaying the signal.

Because anomalies typically manifest as temporal ranges, we complement with event-level metrics. For each true anomalous interval $[t_a^{(i)}, t_b^{(i)}]$, detection latency is defined as $\min\{t \geq t_a^{(i)} : \hat{y}_t = 1\} - t_a^{(i)}$; we also quantify coverage as the fraction of windows in the interval with $\hat{y}_t = 1$, and the event hit rate, which declares an event detected if at least one activation occurs within the interval. This range-based set yields a more faithful picture for time series, where both timeliness and extent of detection matter. These event metrics are obtained via a post-processing tool that takes the offline inference log (errors and scores) and produces a per-event table with latency, coverage, and hit indicator; a summary with means/medians of these quantities alongside window-level metrics; and the window-level confusion matrix and derivatives. Operational details are provided in Algorithm 11 in Appendix A.

The binary decision is generated using the same logic as in deployment. If the file includes a precomputed decision column (`alert_best_f1`) and its use is enabled, we adopt it directly; otherwise, we threshold the combined `score` with a fixed threshold t_h . On top of this raw decision we apply a persistence rule of length K (incrementing a counter on positive decisions and resetting on negatives), emitting an activation when the counter reaches K . This emulation preserves the temporal dynamics of the online stage and reduces isolated triggers.

The operating point used for reporting is fixed according to the calibration criterion: among candidates, we choose the one that maximizes sensitivity subject to a pre-specified FPR bound; if no threshold satisfies the bound, we select the minimum-FPR threshold with ties broken by sensitivity. This scheme is equivalent to choosing a point on the ROC curve under an explicit Type I error constraint, while F_1 summarizes the precision–recall trade-off at the chosen point. All metrics are reported accounting for the temporal shift induced by the recurrent model’s window length, so that each decision $\hat{y}_t$ is compared to its corresponding label.

5.2 Results

5.2.1 Offline Validation and Operating Point Selection

Using the inference flow from §4.11.3, we first ran the AE-LSTM pipeline on the *validation subset* (held out from training) to generate the error logs used for calibration. On this validation split, the grid-based procedure from §4.11.4 was applied, sweeping over weights (α, β) , persistence K , and thresholds t_h (via percentiles) to maximize *recall* under the restriction $\text{FPR} \leq 0.10$ (ties broken by F_1 and then FPR). The selected operating point on validation is summarized in Table 6.

Table 6: Selected operating point in *offline* calibration.

Parameter	α^*	β^*	K^*	t_h^*	Prevalence π	Window T
Value	1.1	1.4	3	15.4413	0.098	20

For context, Table 7 shows the best triplet (α, β, t_h) for each K satisfying $\text{FPR} \leq 0.10$. $K=3$ matches the maximum *recall* (0.999) and, within that maximum, provides the best F_1 with slightly lower FPR than $K=2$; $K=4$ reduces FPR further at the cost of a small recall drop.

Table 7: Best combination by persistence K under $\text{FPR} \leq 0.10$ (*offline*).

K	α	β	t_h	Precision	Recall	F1	FPR
1	0.500	2.000	25.385444	0.491	0.880	0.630	0.099
2	0.500	2.000	19.219507	0.526	0.999	0.689	0.098
3 [†]	1.100	1.400	15.441346	0.529	0.999	0.692	0.097
4	1.100	1.400	15.441346	0.537	0.993	0.697	0.093
5	1.500	1.000	12.477342	0.519	0.992	0.682	0.100
6	1.500	0.800	10.423824	0.522	0.992	0.684	0.099

[†] Reported point (Table 6). Criterion: maximize *recall* under $\text{FPR} \leq 0.10$; tie-breaking by F_1 then FPR.

To visualize the joint effect of (α, β) and persistence K under the restriction $\text{FPR} \leq 0.10$, Fig. 16 shows heatmaps of F_1 . With $K=1$ performance degrades except in regions with high β ; moving to $K=2$ reveals a broad plateau of high F_1 ; $K=3$ maintains maximum *recall* and yields the best F_1 within the bound; and $K=4$ reduces FPR at the cost of a small drop in *recall*. The cross marks the selected point $(\alpha^*, \beta^*) = (1.1, 1.4)$.

5.2.2 Event-Level Results

Event-level evaluation shows complete detection of anomalous episodes (event hit rate = 1.000). Detection latency is consistently below one analysis window, with triggers occurring close to the annotated onset of each anomalous interval once the persistence rule (K consecutive exceedances) is satisfied. Coverage remains high across episodes,

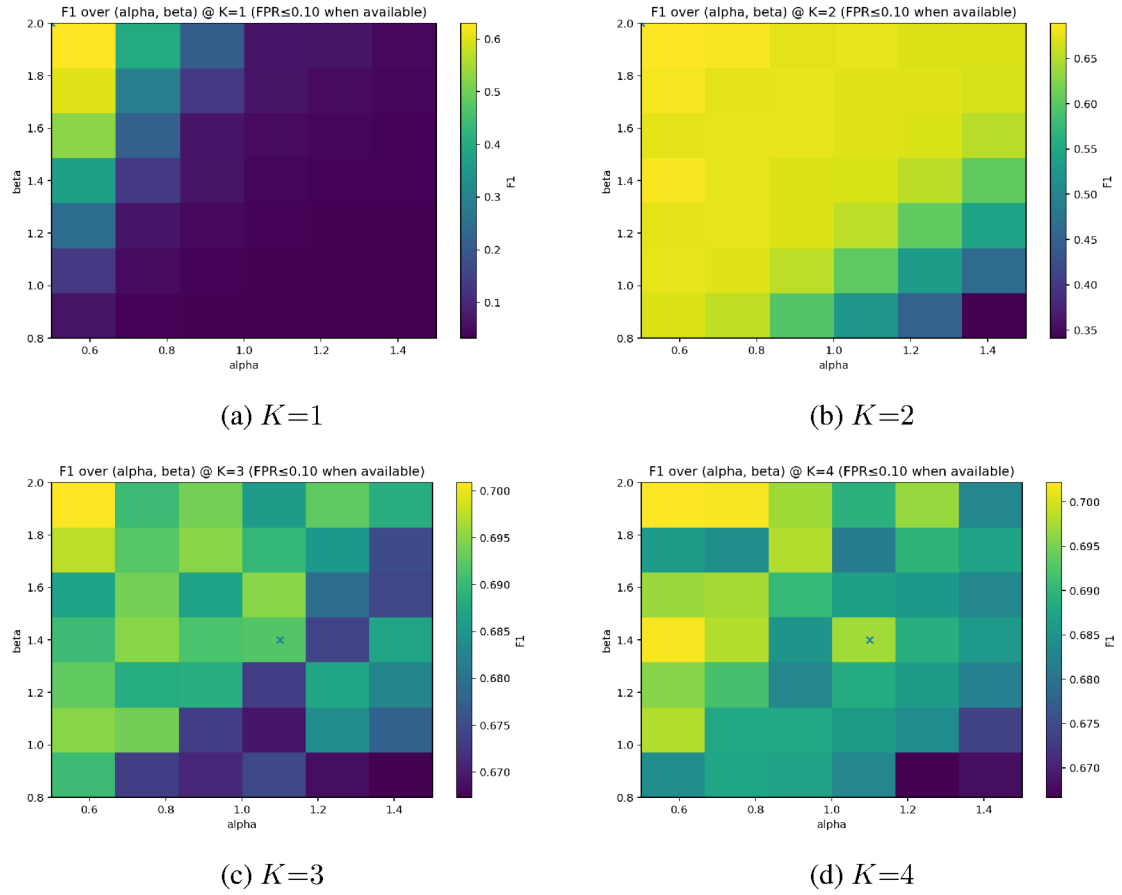


Figure 16: F_1 maps over (α, β) for different values of K under $\text{FPR} \leq 0.10$ (when available). The cross indicates $(\alpha^*, \beta^*) = (1.1, 1.4)$.

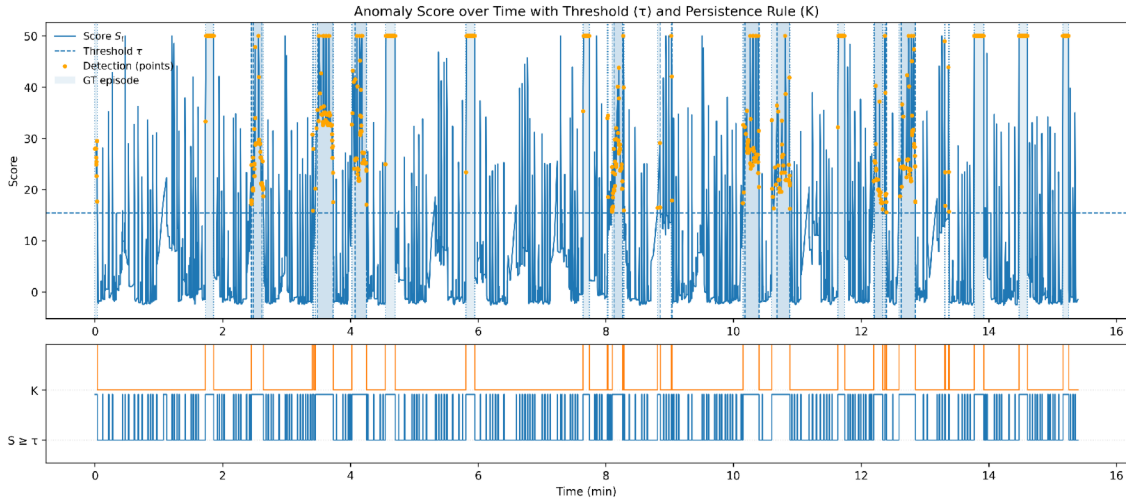


Figure 17: Time series of the anomaly score S_t with threshold τ and K -persistence. Top: raw crossings $S_t \geq \tau$ (points) and GT episodes (shaded). Bottom: consolidated alarms after requiring K consecutive exceedances.

with occasional short gaps in long intervals due to score oscillations around the threshold. Table 8 summarizes these results, confirming that the proposed scheme not only detects all relevant anomalies but does so with rapid and continuous activation.

Figure 17 shows the effect of the persistence rule (K): isolated crossings ($S_t \geq \tau$) are filtered out, whereas GT episodes consolidate into contiguous alarms. The bottom panel highlights the resulting suppression of spurious activations and the short confirmation delay induced by K . This qualitative view complements the event-level metrics in Table 8.

5.3 Discussion

5.3.1 Operating Point and Deployment Considerations

The operating point $(\alpha^*, \beta^*, K^*, t_h^*)$ achieves high sensitivity with controlled FPR, appropriate for online monitoring when the cost of missing events outweighs that of handling false alarms. If the operational cost of false positives is particularly high, the configuration with $K=4$ (Table 7) provides a lower FPR at the expense of a slight recall

Table 8: Event-level results (*offline* evaluation).

Metric	Value	Note
True events	11	—
Event hit rate	1.000	All detected
Latency (median)	0 windows	≈ 0 ms @ 20 Hz
Latency (mean)	0.27 windows	≈ 13.5 ms @ 20 Hz
Coverage (mean)	0.908	Fraction of anomalous interval
Coverage (median)	1.000	Continuous detection in $\geq 50\%$ of events

decrease. In deployment, alarm stability can be reinforced by adding hysteresis or debounce to the output, defining regime-specific thresholds (e.g., depending on speed/turn rate or entropy level), and monitoring nominal drift to anticipate recalibrations.

5.3.2 Limitations and Validity

Results are based on traces with a finite set of regimes and prevalence; scenarios with sustained operational changes or different environmental conditions may require retuning t_h and K or adopting contextual thresholds. The *offline* evaluation reproduces the decision logic but does not capture system load effects and real-time latencies; measuring computational budget per window and inference latency on target hardware is a natural extension.

Still, the mean activation latency per event (~ 13.5 ms, assuming 20 Hz and unit stride) is consistent with a pipeline capable of operating at 20 Hz without perceptible penalty. These values align with the quantitative evidence in Table 8, which synthesizes full event detection, sub-window latencies, and high coverage.

5.3.3 Implications for Computational Modeling

This study contributes to computational modeling by providing a reproducible and frugal pipeline for *online* anomaly detection:

- A hybrid representation combining lightweight entropy metrics (Shannon, transitions, KDE) with a latent space learned by autoencoder, and
- temporal modeling with LSTM trained solely on nominal data and calibrated *offline* under explicit FPR and latency constraints.

Prior stationarity verification (ADF) and robust normalization (median/MAD) support stable thresholds; the persistence rule K formalizes the sensitivity–false alarm trade-off. Running with default ROS 2 Humble/Fast DDS in Docker/Gazebo establishes a replicable baseline that isolates the model effect from middleware factors. Altogether, the approach illustrates sound model-engineering practices: clear separation between representation learning and dynamics, traceable operating criteria (threshold + persistence), and reporting at both window and event levels, transferable to other time series domains.

5.3.4 Anomalies and Data Injection

The anomaly detection developed in this thesis operates as an effective mechanism for identifying attempts of *data injection* in ROS 2/DDS systems. This attack vector manifests as alterations in critical topics (e.g., `/cmd_vel`, `/odom`), either through bias or saturation of values, manipulation of publication cadence (*jitter*, bursts, silences), or replay/impersonation of messages out of context. Such perturbations impact the statistical regularity of inter-arrival intervals and the consistency between commands and robot

state.

The entropy metrics employed (Shannon, transitions, KDE) are sensitive to changes in distributional shape and temporal variability, thus revealing irregular patterns emerging under injection. Complementarily, the *autoencoder* and *LSTM* profile nominal dynamics and expose deviations through significant increases in reconstruction and prediction errors. The robust combination of both scores, normalized via median/MAD and subjected to a persistence rule K , discriminates between normal fluctuations and persistent anomalies associated with an attack.

Consequently, even without inspecting network traffic directly, the system infers the presence of data injection from its footprint in onboard telemetry. This approach strengthens ROS 2 operational security by detecting attacks that degrade cadence, coherence, or signal level, with low latency and explicit false positive control (see Fig. 18).

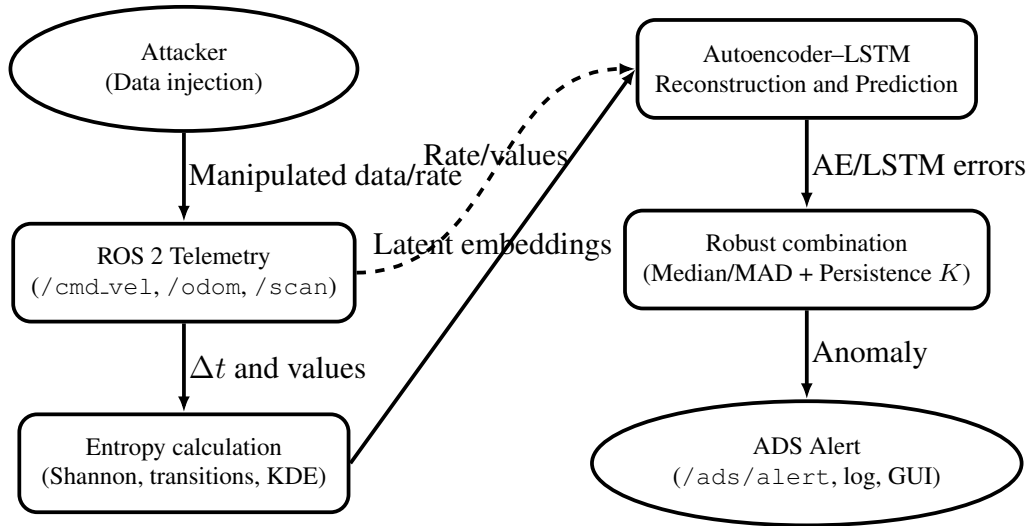


Figure 18: Two-column vertical diagram: the attacker alters data or rate; telemetry affects entropy values; the Autoencoder-LSTM scheme raises the robust score with persistence K and issues the ADS alert.

5.3.5 Executive Summary

Table 9: Executive summary of the *offline* evaluation.

Aspect	Summary
Evaluation objective	Autoencoder–LSTM scheme with robust scaling (median/MAD), fixed threshold and temporal persistence K ; metrics reported at both <i>window</i> and <i>event</i> level.
Operational configuration (selected)	$(\alpha^*, \beta^*, K^*, t_h^*) = (1.1, 1.4, 3, 15.4413)$; prevalence $\pi \approx 0.098$; window length $T = 20$ (Tab. 6).
Window-level metrics	$Recall = 0.999$, $FPR = 0.097$, precision = 0.529, $F1 = 0.692$ (Tab. 7).
Event-level metrics	$Event\ hit\ rate = 1.000$; median latency = 0 windows and mean = 0.27 windows (≈ 13.5 ms at 20 Hz); mean coverage = 0.908 and median = 1.000.
Signal stability	Stable entropies under nominal operation; ADF rejects unit root for Shannon, transitions, and KDE (Tabs. 4 and 5).
Alternative if FP is critical	$K=4$ reduces FPR to 0.093 with $recall = 0.993$ and $F1 = 0.697$ (Tab. 7).
Deployment recommendations	Add hysteresis/ <i>debounce</i> , regime-specific thresholds (e.g., by speed/turn or entropy level), and monitoring of nominal drift. Default ROS 2/Fast DDS setup; suitable for 20 Hz.

6 CONSIDERATIONS AND FUTURE WORK

We synthesize conclusions and contributions to computational modeling, acknowledge limitations, and propose future work directions oriented toward online deployment, hardware validation, and dynamic adaptation.

The empirical scope of this thesis is limited to a controlled and reproducible scenario: simulation in Gazebo with a TurtleBot3, execution of all nodes inside Docker containers, and an exclusive focus on internal telemetry (without instrumenting RTPS/UDP traffic). This configuration reduces confounding variables from the middleware and environment, and enhances traceability of results and artifacts (datasets, seeds, models) for independent replication.

Regarding signal design, entropy metrics were computed online over sliding windows of fixed size $W=100$ samples, with per-sample update (new value in, oldest value out) and calculation of Shannon, transition, and KDE entropies. This choice balances sensitivity to regime changes with stability of the estimate, ensuring that the analysis reflects the most recent regime without dragging past transients.

From the modeling perspective, a deviation-based architecture was adopted, trained solely on nominal data: an autoencoder (AE) that learns a stable latent representation, and an LSTM that forecasts its dynamics one step ahead. This combination was chosen for its complementarity (reconstruction in the observation space and temporal coherence in the latent space) and its suitability for multivariate telemetry in robotic systems.

Offline calibration of the operating point was carried out in the error space (prediction and reconstruction), with robust standardization (median/MAD) and a persistence rule K . Grid search yielded the quartet $(\alpha^*, \beta^*, K^*, t_h^*) = (1.1, 1.4, 3, 15.4413)$, with prevalence $\pi \approx 0.098$ and $T=20$, maximizing sensitivity under the constraint $\text{FPR} \leq 0.10$. Results show near-unit recall ($\text{TPR} = 0.999$) with a single omission, $\text{FPR} \approx 0.097$, and explicit reporting of precision and F_1 to avoid imbalance bias. At the event level, evaluation achieved *event hit rate* = 1.000 and average latency ~ 0.27 windows (~ 13.5 ms at 20 Hz), confirming the adequacy of the scheme for online monitoring.

Operationally, the choice of $K=3$ prioritizes sensitivity while keeping FPR bounded; if the cost of false alarms were critical, $K=4$ offers a slightly lower FPR at the expense

of a small recall reduction. This trade-off was quantified in the persistence sweep.

Finally, the assumptions and limits of external validity should be made explicit. The offline evaluation reproduces the decision logic (fixed threshold + persistence) but does not capture real-time load effects or variations induced by the DDS stack, specific QoS profiles, or DDS Security. Migration to target hardware will require measuring compute budget and inference latency per window, and, under sustained regime changes, anticipating recalibrations (e.g., hysteresis/debounce, contextual thresholds, or adjustment of K). These considerations do not invalidate the results (solidly supported by window- and event-level metrics) but delimit their extrapolation beyond the controlled environment used here.

6.1 Contributions to Computational Modeling

From an empirical standpoint, the experiments carried out on a ROS 2 Humble / Fast DDS setup indicate that the calibrated Autoencoder–LSTM pipeline attains high sensitivity under a bounded false–positive rate and low event-level activation latency at 20 Hz. These results support the feasibility of the proposal for *online* monitoring of robotic telemetry under realistic middleware configurations.

Beyond these performance figures, the main contribution of this work is an operational computational model of anomaly detection in robotic and cyber–physical systems formulated as deviation from a learned nominal regime. Instead of enumerating attack signatures, the model learns regular system dynamics and interprets anomalies as departures from nominal behavior, quantified through a combination of autoencoder reconstruction error and one–step prediction error in latent space. This yields a parsimonious representation centered on normal dynamics and supports deployment in settings where labeled attacks are limited or unavailable.

Second, the proposed architecture is deliberately composable. Sliding–window entropy descriptors, an overcomplete regularized autoencoder, and an LSTM predictor are integrated into a single pipeline that balances interpretability and expressive power. Entropy metrics and reconstruction errors provide transparent views of signal regularity, while the latent LSTM captures subtler temporal dependencies. The pipeline is designed to be reusable across robotic platforms by reconfiguring the feature mapping and recalibrating the operating point for the target telemetry.

Third, calibration and decision making are modeled as a sequential process with explicit control of statistical behavior. Robust standardization based on the median and MAD, a weighted combination of error terms, and persistence–based alert rules are combined to tune the operating point under an explicit constraint on the false–positive rate at both the window and event level. In this way, the detector is not only a predictive model, but a fully specified decision mechanism whose operating characteristics can be analyzed and

adjusted according to operational requirements.

A further contribution is the explicit treatment of the experimental infrastructure as part of the model. The use of ROS 2 Humble with Fast DDS defaults, Gazebo simulations, and containerized execution (Docker) is documented and integrated into the evaluation pipeline, reducing middleware-related confounding factors and enabling reproducible experiments. This facilitates replication, extension, and porting of the methodology to other platforms and network conditions.

Finally, the framework is connected to broader themes in computational modeling, such as model governance and adaptivity under context change. The work outlines natural extensions toward drift management through continuous recalibration, explicit uncertainty quantification, and sim2real validation under dataset shift. Together, these elements define a reusable modeling pattern for anomaly detection in robotics and, more generally, in cyber-physical systems with multivariate time series, where real-time operation and reliability constraints are central.

6.2 Future Work

The next natural step is to transfer the pipeline from simulated setups to real robotic systems via hardware-in-the-loop and on-robot experiments, quantifying end-to-end latency, jitter, and per-window compute budget under varying loads and ROS 2 topologies. This step will contrast the “temporal floor” observed in simulation with that of the physical platform and verify sustained operation at 20 Hz without degrading sensitivity, considering the factors that affect latency and temporal variability in ROS 2/DDS [50, 38]. In parallel, a factorial sensitivity study of QoS profiles and DDS Security is pertinent: varying *Reliability*, *History/Depth*, *Deadline/Liveliness*, message size/fragmentation, and enabling DDS Security to measure their impact on latency, FPR, and event-level detection latency [22, 36, 50], including robustness tests under misconfigured secure pub/sub [84, 18].

Methodologically, ablation and robustness studies should explore the effects of window size (W), effective sampling frequency, variable selection (kinematic vs. sensor-based), AE latent dimension, and LSTM sequence length (T). To stabilize decisions under noise or near-threshold oscillations, hysteresis or debounce could be added to the output, along with Shewhart/EWMA-type rules applied to the score [56]. In continuous operation, adaptive thresholds (heuristic or RL-based) and drift/change detectors will be essential to adjust t_h and/or K when nominal drift is observed, keeping FPR bounded; adaptive windows and streaming statistics maintenance [34, 85, 11, 17], together with range-based anomaly metrics [77], will be useful.

Finally, coverage should be broadened and shareable resources produced: incorporating functional faults (slippage, actuator saturation), sensor degradations (noise, LiDAR occlu-

sions), injection attacks on `/cmd_vel`, bursts and packet loss, while publishing artifacts (scripts, models, replays) and a labeled dataset with event-level metrics. This practice supports reproducibility and auditable evaluation, consistent with evaluation best practices in related domains [33]. Together, these directions consolidate the transfer of the approach to operational environments and real robotic systems, strengthen its external validity, and deepen its contributions to the computational modeling of robotic systems.

As an immediate next step, we will continue with online validation of the detector and implement real-time inference on ROS 2 (Humble) in Gazebo simulation and, progressively, on a real ROS 2-based mobile robot. Specifically, the pipeline (scaling $\rightarrow$ AE encoder $\rightarrow$ LSTM predictor $\rightarrow$ threshold + persistence) will be integrated as a ROS 2 node that consumes live telemetry and publishes alerts to a dedicated topic, maintaining a 20 Hz target cadence. This stage will include instrumentation of end-to-end latency, jitter, and CPU/GPU usage, incorporation of hysteresis/debounce for signal stabilization, and testing under load and QoS variations. The goal is to close the sim2real loop, from a controlled Gazebo environment to deployment on a real robot and/or robotic system in operational scenarios, while preserving traceability of results.

REFERENCES

- [1] Aartsen, M., Banga, K., Talko, K., Touw, D., Wisman, B., Meïnsma, D., and Björkqvist, M. (2022). Analyzing interoperability and security overhead of ros2 dds middleware. pages 976–981.
- [2] Abokhdair, M. and Baig, Z. (2025). A deep learning-based intrusion detection system for ros 2. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE. Manuscrito aceptado.
- [3] Akbarian, H., Mahgoub, I., and Williams, A. (2023). Autoencoder-lstm algorithm for anomaly detection. pages 1–6.
- [4] Antunes, R. A., Dalmazo, B. L., and Drews, P. L. J. (2022). Detecting data injection attacks in ros systems using machine learning. In *2022 Latin American Robotics Symposium (LARS), Brazilian Symposium on Robotics (SBR) and Workshop on Robotics in Education (WRE)*, pages 1–6. IEEE. Proc. LARS/SBR/WRE 2022.
- [5] Bandt, C. and Pompe, B. (2002). Permutation entropy: A natural complexity measure for time series. *Physical Review Letters*, 88(17):174102.
- [6] Basseville, M. and Nikiforov, I. V. (1993). *Detection of Abrupt Changes: Theory and Application*. Prentice Hall.
- [7] Bédard, C., Lütkebohle, I., and Dagenais, M. (2022). `ros2_tracing`: Multipurpose low-overhead framework for real-time tracing of ROS 2. *IEEE Robotics and Automation Letters*, 7(3):6511–6518.
- [8] Beirlant, J., Dudewicz, E. J., Györfi, L., and van der Meulen, E. C. (1997). Nonparametric entropy estimation: An overview. *International Journal of Mathematical and Statistical Sciences*, 6(1):17–39.
- [9] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

- [10] Berahmand, K., Daneshfar, F., Salehi, E., et al. (2024). Autoencoders and their applications in machine learning: A survey. *Artificial Intelligence Review*, 57:28. <https://doi-org.proxy.timbo.org.uy/10.1007/s10462-023-10662-6>.
- [11] Bifet, A. and Gavalda, R. (2007). Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*, pages 443–448.
- [12] Blázquez-García, A. et al. (2021). A review on outlier/anomaly detection in time series data. *ACM Computing Surveys*, 54(3):1–33.
- [13] Callegari, C., Giordano, S., and Pagano, M. (2017). Entropy-based network anomaly detection. In *2017 International Conference on Computing, Networking and Communications (ICNC)*, pages 334–339. IEEE.
- [14] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58.
- [15] Cheng, Y. X., Harada, K., and Kajita, S. (2019). A latent state-based multimodal execution monitor with anomaly classification for robot manipulation. *Applied Sciences*, 9(6):1072.
- [16] Cover, T. M. and Thomas, J. A. (2006). Elements of information theory, ch. 2 (entropy). https://sgfin.github.io/files/notes/Cover_and_Thomas_ch2_entropy.pdf.
- [17] Datar, M., Gionis, A., Indyk, P., and Motwani, R. (2002). Maintaining stream statistics over sliding windows. In *Proceedings of the 13th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA)*, pages 635–644.
- [18] Deng, G., Xu, G., Zhou, Y., Zhang, T., and Liu, Y. (2022a). On the (in)security of secure ROS 2. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS '22)*, pages 739–753, Los Angeles, CA, USA. ACM.
- [19] Deng, G., Xu, G., Zhou, Y., Zhang, T., and Liu, Y. (2022b). On the (in)security of secure ros2. page 15 pages.
- [20] Dickey, D. A. and Fuller, W. A. (1979). Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, 74(366):427–431.
- [21] Dieber, B., Breiling, B., Taurer, S., Kacianka, S., Rass, S., and Schartner, P. (2017). Security for the robot operating system. In *IEEE IROS*, pages 4477–4484.

- [22] DiLuoffo, V., Michalson, W. R., and Sunar, B. (2018). Robot operating system 2: The need for a holistic security approach to robotic architectures. *International Journal of Advanced Robotic Systems*, 15(3):1–15.
- [23] Enders, W. (2014). *Applied Econometric Time Series*. Wiley, 4 edition.
- [24] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874.
- [25] Freedman, D. and Diaconis, P. (1981). On the histogram as a density estimator: l_2 theory. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 57(4):453–476.
- [26] Gama, J. (2010). *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC.
- [27] Gao, R., Tang, P.-H., Wang, S.-W., Tsai, D.-I., Zhang, Z., and Bian, G. (2023). Ultrasound entropy imaging based on the kernel density estimation: A new approach to hepatic steatosis characterization. *Diagnostics*, 13(24):3646.
- [28] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press, Cambridge, MA.
- [29] Grantham, K. and van den Heuvel, E. R. (2023). Monitoring univariate processes using control charts: some practical issues and advice. https://pure.uva.nl/ws/files/207771767/Monitoring_univariate_processes_using_control_charts_Some_practical_issues_and_advice.pdf.
- [30] Hamilton, J. D. (1994). *Time Series Analysis*. Princeton University Press.
- [31] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507.
- [32] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.
- [33] Hundman, K., Constantinou, V., Laporte, J., Colwell, I., and Soderstrom, T. (2018). Detecting spacecraft anomalies using lstms and nonparametric dynamic thresholding. In *KDD*, pages 387–395.
- [34] Isaac, K. and Sharma, P. (2023). Adaptive thresholding heuristic for kpi anomaly detection. *arXiv preprint arXiv:2308.10504*.
- [35] J., F. S. and C., G. M. (2023). Anomaly detection in network traffic using entropy-based methods. *Issues in Information Systems*, 24(1):82–94.

- [36] Kim, J., Smereka, J. M., Cheung, C., Nepal, S., and Grobler, M. (2018). Security and performance considerations in ros 2: A balancing act. *arXiv preprint arXiv:1809.09566*.
- [37] Koenig, N. and Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2149–2154, Sendai, Japan. IEEE.
- [38] Kronauer, T., Pohlmann, J., Matth  , M., Smejkal, T., and Fettweis, G. (2021). Latency analysis of ROS 2 multi-node systems. *arXiv preprint arXiv:2101.02074*.
- [39] Lacheekhab, F., Benzaoui, M., Tadjer, S. A., Bensmaine, A., and Hamma, H. (2024). Lstm-autoencoder deep learning model for anomaly detection in electric motor. *Energies*, 17(10).
- [40] Lakhina, A., Crovella, M., and Diot, C. (2004). Mining anomalies using traffic feature distributions. In *ACM SIGCOMM*, pages 217–228.
- [41] Lee, Y., Park, C., Kim, N., Ahn, J., and Jeong, J. (2024). Lstm-autoencoder based anomaly detection using vibration data of wind turbines. *Sensors*, 24(9).
- [42] Lin, S., Clark, R., Birke, R., Sch  nborn, S., Trigoni, N., and Roberts, S. (2020). Anomaly detection for time series using vae-lstm hybrid model. pages 4322–4326.
- [43] Lindstrom, M. R., Jung, H., and Larocque, D. (2020). Functional kernel density estimation: Point and fourier approaches to time series anomaly detection.
- [44] Liu, X., Wang, Y., et al. (2016). Increment entropy as a measure of complexity for time series. *Entropy*, 18(1):38.
- [45] Lu, T., Wang, L., and Zhao, X. (2023). Review of anomaly detection algorithms for data streams. *Applied Sciences*, 13:6353. <https://doi.org/10.3390/app13106353>.
- [46] Luo, R. and Lee, T.-T. (2015). Model-free robot anomaly detection. In *Proceedings of the International Symposium on Robotics*.
- [47] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. (2022). Robot operating system 2: Design architecture and uses in the wild. *Science Robotics*, 7(66):eabm6074.
- [48] Malhotra, P. et al. (2015). Long short term memory networks for anomaly detection in time series. In *ESANN*.

- [49] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., and Shroff, G. (2016). Lstm-based encoder-decoder for multi-sensor anomaly detection. *arXiv preprint arXiv:1607.00148*.
- [50] Maruyama, Y., Kato, S., and Azumi, T. (2016). Exploring the performance of ROS 2. In *Proceedings of the 13th International Conference on Embedded Software (EMSOFT)*, pages 1–10. ACM.
- [51] Myers, A. et al. (2025). Entropic analysis of time series through kernel density estimation. <https://arxiv.labs.arxiv.org/html/2503.18916>.
- [52] Nam, H., Jeong, Y.-K., and Park, J. W. (2020). An anomaly detection scheme based on lstm autoencoder for energy management. pages 1445–1447.
- [53] Nardone, P. (2014). Entropy of difference. *arXiv preprint arXiv:1411.0506*.
- [54] Nelson, L. S. (1984). The shewhart control chart—tests for special causes. *Journal of Quality Technology*, 16(4):237–239.
- [55] Ng, A. (2011). Sparse autoencoder. Lecture notes, CS294A (Unsupervised Deep Learning), Stanford University. Updated Jan 11, 2011. Accessed: 2025-09-17.
- [56] NIST/SEMATECH (2024). Shewhart control chart. <https://www.itl.nist.gov/div898/handbook/mpc/section2/mpc221.htm>. NIST/SEMATECH e-Handbook of Statistical Methods.
- [57] Olshausen, B. A. and Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23):3311–3325.
- [58] Open Robotics and ROS 2 Contributors (2022). Quality of service (qos) settings in ros 2. ROS 2 Documentation — Humble. Accedido: 2025-09-23.
- [59] Park, D., Hoshi, Y., and Kemp, C. C. (2017). A multimodal anomaly detector for robot-assisted feeding using an lstm-based variational autoencoder. *arXiv preprint arXiv:1711.00614*.
- [60] Pawlikowski, K. et al. (2007). Steady state models and the initial transient. Technical report, University of Warwick.
- [61] Ptaszynski, P., Sikora, M., Bozek, P., and Wojciechowski, K. (2024). A real-time anomaly detection method for robots based on multivariate time series forecasting. *arXiv preprint*.
- [62] Radivilova, T., Kirichenko, L., and Alghawli, A. S. (2019). Entropy analysis method for attacks detection. In *2019 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*, pages 441–446. IEEE.

- [63] Rahimi, A. and Recht, B. (2007). Random features for large-scale kernel machines. *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1177–1184.
- [64] Ribeiro, M., Lacerda, G., and Vieira, M. (2017). Entropy-based anomaly detection in large-scale systems. *arXiv preprint arXiv:1705.04779*.
- [65] Rifai, S., Vincent, P., Muller, X., Glorot, X., and Bengio, Y. (2011). Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, pages 833–840.
- [66] Robotics, O. (2019). Ros 2 design. Sitio web oficial. Disponible en: <https://design.ros2.org/>.
- [67] ROS 2 Contributors (2022). rclcpp::qos class reference. ROS 2 C++ API — Humble. Accedido: 2025-09-23.
- [68] Rousseeuw, P. J. and Croux, C. (1993). Alternatives to the median absolute deviation. *Journal of the American Statistical Association*, 88(424):1273–1283.
- [69] Said, S. E. and Dickey, D. A. (1984). Testing for unit roots in autoregressive-moving average models of unknown order. *Biometrika*, 71(3):599–607.
- [70] Sakurada, M. and Yairi, T. (2014). Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the 2nd Workshop on Machine Learning for Sensory Data Analysis (MLSDA) at ACM RecSys*, pages 4–11.
- [71] Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.
- [72] Schwert, G. W. (1989). Tests for unit roots: A monte carlo investigation. *Journal of Business & Economic Statistics*, 7(2):147–159.
- [73] Scott, D. W. (1979). On optimal and data-based histograms. *Biometrika*, 66(3):605–610.
- [74] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 623–656.
- [75] Silverman, B. W. (1986). *Density Estimation for Statistics and Data Analysis*. Chapman & Hall.
- [76] Song, Y., Hyun, S., and Cheong, Y.-G. (2021). Analysis of autoencoders for network intrusion detection. *Sensors*, 21(13):4294. <https://doi.org/10.3390/s21134294>.

- [77] Tatbul, N., Lee, T. J., Zdonik, S., Alam, M., and Gottschlich, J. (2018). Precision and recall for time series. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*. Curran Associates, Inc.
- [78] Unakafov, A. M. and Keller, K. (2018). Change-point detection using the conditional entropy of ordinal patterns. *Entropy*, 20(9):709.
- [79] Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th International Conference on Machine Learning (ICML)*, pages 1096–1103. ACM.
- [80] Wand, M. P. (1994). Fast computation of multivariate kernel estimators. *Journal of Computational and Graphical Statistics*, 3(4):433–445.
- [81] Williams, M. O., Kevrekidis, I. G., and Rowley, C. W. (2015). A data-driven approximation of the koopman operator: Extending dynamic mode decomposition. *Journal of Nonlinear Science*, 25:1307–1346.
- [82] Yan, Y., Qi, L., Wang, J., Lin, Y., and Chen, L. (2020). A network intrusion detection method based on stacked autoencoder and lstm. pages 1–6.
- [83] Yang, S. (2022). Anomaly traffic detection based on lstm. pages 667–670.
- [84] Yang, S. et al. (2024). Formal analysis and detection for ros2 communication security vulnerability. *Electronics*, 13(9):1762.
- [85] Yang, W., Zhao, C., and Ma, J. (2023). Dynamic thresholding based on reinforcement learning for anomaly detection. *arXiv preprint arXiv:2312.01488*.
- [86] Ziviani, A., Monsore, J., Velloso, P., Rodrigues, H., and Gomes, D. (2007). Network anomaly detection using nonextensive entropy. *IEEE Communications Letters*, 11(12):1034–1036.

Appendices

A ALGORITHMS

Algorithm 1: Sliding-window update and entropy computation in `DataCollectorNode`

Input: Window size $W = 100$; interval buffer $\mathcal{B}$ (empty); new timestamp t_k ;
previous timestamp t_{k-1}

Output: Entropies $\{H_{\text{Shannon}}, H_{\text{Trans}}, H_{\text{KDE}}\}$ computed on $\mathcal{B}$

```
// 1) Compute the new time interval
1  $\Delta t \leftarrow t_k - t_{k-1}$ ;
// 2) Update the sliding window
2 Append  $\Delta t$  to  $\mathcal{B}$ ;
3 if  $|\mathcal{B}| > W$  then
4   Remove the oldest element from  $\mathcal{B}$ ;
// 3) Shannon entropy (discrete, via histogram on  $\mathcal{B}$ ,
//    base 2)
5 Build a histogram of  $\mathcal{B}$  with suitable bins;
6 Obtain probabilities  $\{p_i\}$  and compute  $H_{\text{Shannon}} = -\sum_i p_i \log_2(p_i + \varepsilon)$  with
    $\varepsilon = 10^{-9}$ ;
// 4) Transition entropy (first order, on discretized
//    states)
7 Discretize  $\mathcal{B}$  into states  $\{s_j\}$ ;
8 Estimate the transition matrix  $P(s_j \rightarrow s_{j'})$ ;
9 Compute  $H_{\text{Trans}} = -\sum_j \sum_{j'} P_{jj'} \log_2(P_{jj'} + \varepsilon)$ ;
// 5) KDE-based (differential) entropy
10 Estimate the density  $f(x)$  on  $\mathcal{B}$  via KDE;
11 Approximate  $H_{\text{KDE}} = -\int f(x) \log f(x) dx$ ; convert to bits by dividing por  $\ln 2$ ;
12 return  $\{H_{\text{Shannon}}, H_{\text{Trans}}, H_{\text{KDE}}\}$ ;
```

Algorithm 2: Shannon entropy computation

Input: Discrete sequence $V = \{v_1, \dots, v_n\}$
Output: Shannon entropy H (bits)

```

1 if  $|V| < 2$  then
2   return 0.0;                                /* Insufficient variation */
  // 1) Frequency counts
3 Build a frequency dictionary  $\text{freq}$  where  $\text{freq}[x]$  is the count of  $x$  in  $V$ ;
  // 2) Empirical probabilities
4  $\text{total} \leftarrow |V|$ ;
5 Construct  $P = \left\{ \frac{\text{freq}[x]}{\text{total}} \mid x \in \text{freq} \right\}$ ;
  // 3) Numerically stable entropy (base 2)
6  $\varepsilon \leftarrow 10^{-9}$ ;
7  $H \leftarrow - \sum_{p \in P} p \cdot \log_2(p + \varepsilon)$ ;
8 return  $H$ ;
```

Algorithm 3: Transition entropy via first differences (proxy)

Input: Sequence $V = \{v_1, v_2, \dots, v_n\}$
Output: Proxy H_{cond} (bits)

```

1 if  $|V| < 2$  then
2   return 0.0;                                /* Not enough data */
  // 1) Consecutive differences
3  $\Delta V \leftarrow \{v_{i+1} - v_i \mid i = 1, \dots, n-1\}$ ;
  // 2) Shannon entropy of differences (base 2)
4  $H_{\text{cond}} \leftarrow \text{ShannonEntropy}(\Delta V)$ ;
5 return  $H_{\text{cond}}$ ;
```

Algorithm 4: KDE differential entropy on a sliding window (plug-in, LOO)

Input: Sequence V ; window size W ; bandwidth h (or Scott/Silverman); base = bits

Output: H_{KDE} on the last window

```

1 if  $|V| < 2$  then
2   return 0.0 ;                               /* Insufficient data */
3 1) Window  $m \leftarrow \min(W, |V|)$ ;  $X \in \mathbb{R}^{m \times 1}$  last  $m$  values.;
4 2) KDE Fit Gaussian KDE  $\hat{f}_h$  on  $X$ .;
5 3) LOO log-density For each  $x_i$ :  $\ell_i \leftarrow \log \hat{f}_{-i}(x_i)$  (fallback  $\log \hat{f}(x_i)$ );
6 4) Entropy  $h_{\text{nats}} \leftarrow -\frac{1}{m} \sum_{i=1}^m \ell_i$ ;  $H_{\text{KDE}} \leftarrow h_{\text{nats}} / \ln 2$  (bits);
7 return  $H_{\text{KDE}}$ ;

```

Algorithm 5: Anomaly activation with random duration and logging

Input: Current anomaly mode $\text{mode} \in \{\text{detection}, \text{training}, \text{attack}\}$;
anomaly type

Output: Anomaly activated, logged, and simulation started

```

// 1) Set duration according to mode
1 if  $\text{mode} = \text{detection}$  then
2    $\text{duration} \leftarrow 3.0 + \mathcal{U}(0, 2.0)$ 
3 else if  $\text{mode} = \text{training}$  then
4    $\text{duration} \leftarrow 5.0 + \mathcal{U}(0, 5.0)$ 
5 else if  $\text{mode} = \text{attack}$  then
6    $\text{duration} \leftarrow 15.0 + \mathcal{U}(0, 15.0)$ 
7 else
8    $\text{duration} \leftarrow 5.0$ ;                               /* Default */
// 2) Update state & timing
9  $\text{anomaly\_active} \leftarrow \text{true}$ ;  $\text{anomaly\_end\_time}$ 
    $\leftarrow \text{now}() + \text{duration}$ ;  $\text{anomaly\_elapsed\_time} += \text{duration}$ ;
    $\text{attack\_type\_current} \leftarrow \text{type}$ ;
// 3) Publish and log
10 Publish attack type; build label = anomaly_+type; publish motion label;
     $\text{log}(\text{type}, \text{duration})$ ;
// 4) Invoke simulation
11 Call routine for type with duration;

```

Algorithm 6: Simulation of the *stop-and-go* anomaly

Input: Total duration T
Output: Sequence of `Twist` with intermittent stops/forward motions

```

1  $t_{\text{end}} \leftarrow \text{now}() + T$ ;
2 while  $\text{now}() < t_{\text{end}}$  and  $\text{rcldpy.ok}()$  do
    // Choose state
3   if  $\mathcal{U}(0, 1) < 0.5$  then
4      $v \leftarrow 0.0$ ;
5      $w \leftarrow 0.0$ ;                                /* Full stop */
6   else
7      $v \leftarrow 0.4$ ;
8      $w \sim \mathcal{U}(-0.2, 0.2)$ ;                        /* Forward with oscillation */
    // Publish
9   Publish Twist with linear.x = v, angular.z = w; label
    anomaly_stop-and-go;
    // Wait
10  Sleep  $\tau \in \{0.3, 0.5, 1.0\}$  s (random);
  
```

Algorithm 7: Logging anomalies to a CSV file

Input: Anomaly type, duration

Output: Entry appended to a CSV log

```

1  $t_{\text{start}} \leftarrow \text{current time}$ ;  $t_{\text{end}} \leftarrow t_{\text{start}} + \text{duration}$ ;
2  $\text{start\_str} \leftarrow \text{format } t_{\text{start}} \text{ as YYYY-MM-DD HH:MM:SS}$ ;  $\text{end\_str} \leftarrow$ 
    $\text{format } t_{\text{end}} \text{ likewise}$ ;
3 Open CSV in append mode;;
4 Write row: session id,  $t_{\text{start}}$ ,  $\text{start\_str}$ , type,  $t_{\text{end}}$ ,  $\text{end\_str}$ , anomaly mode;;
5 Close CSV;
  
```

Algorithm 8: Training and export of the overcomplete autoencoder (8→10)

Input: CSV with nominal data (8 features); list `FEATURES`; 80/20 temporal split; `StandardScaler` (fit on train only);
 AE MLP: encoder $8 \rightarrow 128 \rightarrow 64 \rightarrow 10$ (ReLU, optional dropout), symmetric decoder;
 Adam ($\eta=10^{-3}$, $\text{wd}=10^{-5}$); `ReduceLROnPlateau` (factor 0.5, patience 8); early stopping (patience 20); max 150 epochs;
 $K_\sigma=3$ for $\tau=\mu+K_\sigma\sigma$.
Output: Trained AE; serialized encoder; scaler; curves/CSV (MSE/MAE per epoch); threshold τ (valid);
 latent embeddings (train/val/all) preserving temporal order.

- 1 **Load & clean:** read CSV; verify `FEATURES`; drop $\pm\infty/\text{NaN}$; align `Timestamp` if exists.;
 - 2 **Temporal split:** 80%/20% preserving order.;
 - 3 **Scaling:** fit scaler on train; transform train/valid; save scaler.;
 - 4 **Training:** init AE; Adam; scheduler; train ≤ 150 epochs with early stopping; save best.;
 - 5 **Threshold (valid):** reconstruct valid; get per-row MSE; $\mu, \sigma; \tau = \mu + K_\sigma\sigma$; save JSON.;
 - 6 **Export:** save AE/encoder; export graphs/metrics.;
 - 7 **Embeddings:** encode train/valid/all; save .npy/.csv (with `Timestamp`).;
-

Algorithm 9: Training the LSTM predictor (residual, one step ahead)

Input: $z_t \in \mathbb{R}^D$ (train/val); seq length T ; hyperparams $(H, \eta, B, E, \text{pat}, \text{clip}, \lambda)$
Output: F_θ ; latent scaler; error thresholds (valid)

- 1 **Load & split:** load $Z_{\text{train}}, Z_{\text{val}}$ (temporal order);
 - 2 **Scaling:** fit `StandardScaler` on Z_{train} only; transform Z_{val} ;
 - 3 **Sequences:** $\mathbf{X}_t = (z_{t-T+1}, \dots, z_t)$, target $\mathbf{Y}_t = z_{t+1}$;
 - 4 **Model:** 1-layer LSTM with H units + linear head; residual $\hat{z}_{t+1} = z_t + \Delta_\theta(\mathbf{X}_t)$;
 - 5 **Training:** SmoothL1 + Adam (η, λ); `ReduceLROnPlateau`; early stopping (pat); grad clipping (norm $\leq \text{clip}$); track MSE/MAE;
 - 6 **Selection:** keep lowest valid MSE;
 - 7 **Thresholds:** on valid, $e = \|z_{t+1} - \hat{z}_{t+1}\|_2$; derive $\mu, \sigma, \mu+3\sigma, p99, p99.7$;
 - 8 **Outputs:** export predictor (incl. serialized), scaler, curves, metrics, metadata;
-

Algorithm 10: Offline inference and evaluation (operational overview)

Input:Evaluation CSV with raw *features* and optional Anomaly labels.Artifacts: `scaler.pkl`, `encoder.pth`, `autoencoder_full.pth` or`autoencoder_state.pth`, `scaler_lstm.pkl`, `lstm_predictor.ts.pt`,`lstm_pred_threshold.json` (contains `seq_len`, `latent_dim`, optional α, β, K).**Output:** `errors_timeseries.csv`, `threshold_sweep.csv`, `hist_error.png`,`timeseries_error.png`, `summary.json`.**1. Load and validate**2 Read CSV $\rightarrow df$; verify required feature columns; drop rows with NaN/Inf in features.

3 If Timestamp exists, sort by Timestamp and keep alignment.

2. AE embeddings5 Load `scaler.pkl` and transform raw features $\rightarrow X$.6 Load `encoder.pth`; compute embeddings $z_t \in \mathbb{R}^D$ (preserving order).**3. LSTM prediction & error**8 Load `scaler_lstm.pkl`; transform $z_t \rightarrow \tilde{z}_t$.9 Read `seq_len = T` and `latent_dim = D` from `lstm_pred_threshold.json`.10 $\{X_k, Y_k\} \leftarrow \text{FormSequences}(\tilde{z}_{1:N}, T)$, with $X_k = (\tilde{z}_{k-T+1}, \dots, \tilde{z}_k)$ and $Y_k = \tilde{z}_{k+1}$.11 Load `lstm_predictor.ts.pt`; infer residual $\hat{\Delta}_k$ and set $\hat{z}_{k+1} = z_k + \hat{\Delta}_k$ (residual update).12 Compute $e_t^{\text{LSTM}} = \|\hat{z}_t - z_t\|_2$ for $t = T + 1, \dots, N$.**4. AE reconstruction error**14 If `autoencoder_full.pth` (or state) available, load AE; reconstruct $\hat{x}_t$ from z_t or directly from x_t (depending on the architecture).15 Compute $e_t^{\text{AE}} = \text{RMSE}(x_t, \hat{x}_t)$ and align with $t = T + 1, \dots, N$.16 If AE not available, set $e_t^{\text{AE}} \leftarrow \emptyset$ and use only LSTM error.**5. Robust standardization & combined score**18 From `lstm_pred_threshold.json` read optional weights (α, β) (default $\alpha=1, \beta=1$) and persistence K (default $K=1$).19 $\tilde{e}_t^{\text{LSTM}} \leftarrow \text{RobustStandardize}(e_t^{\text{LSTM}})$;20 If $e_t^{\text{AE}} \neq \emptyset$: $\tilde{e}_t^{\text{AE}} \leftarrow \text{RobustStandardize}(e_t^{\text{AE}})$.21 Define $S_t = \alpha \tilde{e}_t^{\text{LSTM}} + \beta \tilde{e}_t^{\text{AE}}$ (or $S_t = \tilde{e}_t^{\text{LSTM}}$ if AE not available).**6. Threshold sweep and persistence**23 Build grid $\Theta \leftarrow \text{PercentileGrid}(S_t, \{80, 85, \dots, 99.9\})$ (adjustable).24 For each $\theta \in \Theta$: raw decisions $b_t = \mathbb{I}\{S_t > \theta\}$;25 Apply persistence: $y_t^{(\theta)} \leftarrow \text{ApplyPersistence}(b_t, K)$.26 If Anomaly exists in df : compute metrics per θ (precision, recall, F1; AUROC si aplica).27 Select θ^* por mejor F1 (o FPR objetivo) y guardar.**7. Outputs**29 Write `errors_timeseries.csv`: Timestamp (if any), e_t^{LSTM} , e_t^{AE} (if any), S_t , $y_t^{(\theta^*)}$.30 Write `threshold_sweep.csv`: θ , metrics, prevalences.31 Save `hist_error.png` (hist of S_t) and `timeseries_error.png` (time series with θ^*).32 Write `summary.json`: θ^* , K , (α, β) , `seq_len`, `latent_dim`, event/alert counts, global metrics.

Algorithm 10: Offline inference and evaluation (continued)

```

33 Auxiliary functions
34 Func FormSequences ( $z_{1:N}, T$ )
35    $\mathcal{X} \leftarrow [], \mathcal{Y} \leftarrow []$ 
36   for  $k = T$  to  $N - 1$  do
37     Append ( $z_{k-T+1:k}$ ) to  $\mathcal{X}$ ; Append ( $z_{k+1}$ ) to  $\mathcal{Y}$ 
38   end
39   return ( $\mathcal{X}, \mathcal{Y}$ )
40 Func RMSE ( $x, \hat{x}$ )
41   return  $\sqrt{\frac{1}{d} \sum_{i=1}^d (x_i - \hat{x}_i)^2}$ 
42 Func RobustStandardize ( $e_{1:M}$ )
43    $m \leftarrow \text{median}(e)$ ;  $\text{mad} \leftarrow \text{median}(|e - m|) + 10^{-12}$ 
44   return  $(e - m) / \text{mad}$ 
45 Func PercentileGrid ( $v, P$ )
46   return  $\{\text{percentile}(v, p) : p \in P\}$ 
47 Func ApplyPersistence ( $b_{1:M}, K$ )
48   if  $K \leq 1$  then
49     return  $b$ 
50   end
51    $\text{out} \leftarrow \mathbf{0}, \text{run} \leftarrow 0$ 
52   for  $i = 1$  to  $M$  do
53     if  $b_i = 1$  then
54        $\text{run} \leftarrow \text{run} + 1$ ;  $\text{out}_i \leftarrow \mathbb{I}\{\text{run} \geq K\}$ 
55     else
56        $\text{run} \leftarrow 0$ ;  $\text{out}_i \leftarrow 0$ 
57     end
58   end
59   return  $\text{out}$ 
60

```

Algorithm 11: Computation of event-level metrics from `errors.timeseries.csv`

File `errors.timeseries.csv` with columns:

▷ Anomaly (binary, required)

Input: ▷ `alert_best_f1` or score (one required)

▷ Timestamp (optional)

Parameters: `USE_ALERT_FROM_CSV` (bool), `THRESHOLD` (real), `K_PERSIST` (integer)

Output: `metricas_eventos/event_level_metrics.csv`: per-event metrics

`metricas_eventos/event_summary.json`: event-level aggregates

1. Load and validate

2 Load the CSV into a DataFrame `df`. Verify that Anomaly exists and that either `alert_best_f1` or score is present.

3. Raw decision (0/1)

4 **if** `USE_ALERT_FROM_CSV` is True **and** `alert_best_f1` exists **then**

5 | `raw_dec` $\leftarrow$ `alert_best_f1`

6 | `decision_source` $\leftarrow$ "alert_best_f1"

7 **end**

8 **else**

9 | Require score.

10 | `raw_dec` $\leftarrow$ $\mathbb{I}\{\text{score} > \text{THRESHOLD}\}$

11 | `decision_source` $\leftarrow$ "score $\geq$ THRESHOLD"

12 **end**

13. Persistence

14 `yhat` $\leftarrow$ `ApplyPersistence(raw_dec, K_PERSIST)` // emulates online logic

15. True events (labels)

16 `y_true` $\leftarrow$ `df[Anomaly]`

17 `true_events` $\leftarrow$ `ContiguousRuns(y_true, 1)` // list of (start, end) pairs

18. Timestamps (optional)

19 **if** `Timestamp` $\in$ columns of `df` **then**

20 | `(ts_dt, valid_frac, step_s)` $\leftarrow$ `ParseTSLoose(df[Timestamp])`

21 **end**

22 **else**

23 | `ts_dt` $\leftarrow$ $\emptyset$

24 **end**

25. Per-event metrics

26 `rows` $\leftarrow$ [], `event_id` $\leftarrow$ 0

27 **foreach** $(a, b) \in \text{true_events}$ **do**

28 | `len` $\leftarrow$ `b - a + 1`

29 | `seg_dec` $\leftarrow$ `yhat[a:b]`

30 | `hit` $\leftarrow$ $\bigvee \text{seg_dec}$ // any detection within the event?

31 | `first_idx` $\leftarrow$ None, `lat_win` $\leftarrow$ None, `lat_sec` $\leftarrow$ None

32 **if** `hit` **then**

33 | `rel` $\leftarrow$ `argmin{i : seg_dec[i] = 1}`

34 | `first_idx` $\leftarrow$ `a + rel`

35 | `lat_win` $\leftarrow$ `first_idx - a`

36 | **if** `ts_dt` $\neq \emptyset$ **then**

37 | | `lat_sec` $\leftarrow$ `(ts_dt[first_idx] - ts_dt[a])` in seconds

38 | **end**

39 **end**

40 `coverage` $\leftarrow$ `mean(seg_dec)`

41 Append to `rows`:

| {`event_id`, `a`, `b`, `len`, `hit`, `first_idx`, `lat_win`, `lat_sec`, `coverage`}

42 `event_id` $\leftarrow$ `event_id + 1`

43 **end**

Algorithm 11: Computation of event-level metrics from `errors_timeseries.csv` (continued)

44 7. Persistent outputs

45 Write `event_level_metrics.csv` with rows.

46 Compute aggregates: number of events, hit rate, and summary statistics of `lat_win`, `lat_sec`, and coverage.

47 Build and save `event_summary.json` with: `decision_source`, `THRESHOLD` (if applicable), `K_PERSIST`, timestamp info, and aggregate metrics.

48 Auxiliary functions

49 **Func** `ApplyPersistence` ($b[1:N], k$)

50 **if** $k \leq 1$ **then**

51 **return** b

52 **end**

53 $out \leftarrow \mathbf{0}$, $run \leftarrow 0$

54 **for** $i \leftarrow 1$ **to** N **do**

55 **if** $b[i] = 1$ **then**

56 $run \leftarrow run + 1$; $out[i] \leftarrow \mathbb{I}\{run \geq k\}$

57 **else**

58 $run \leftarrow 0$; $out[i] \leftarrow 0$

59 **end**

60 **end**

61 **return** out

62 **Func** `ContiguousRuns` ($y[1:N], value$)

63 $pairs \leftarrow []$; $s \leftarrow \text{None}$

64 **for** $i \leftarrow 1$ **to** N **do**

65 **if** $y[i] = value$ **and** $s = \text{None}$ **then**

66 $s \leftarrow i$

67 **end**

68 **if** ($y[i] \neq value$ **or** $i = N$) **and** $s \neq \text{None}$ **then**

69 $e \leftarrow i - \mathbb{I}\{y[i] \neq value\}$

70 Append (s, e) to $pairs$; $s \leftarrow \text{None}$

71 **end**

72 **end**

73 **return** $pairs$

74 **Func** `ParseTSLoose` ($Timestamp$)

75 Try `to_datetime`; if valid fraction low, try epoch (s/ms).

76 Return $(ts_dt, valid_frac, median_step_seconds)$.

Algorithm 12: Offline tuning of weights, persistence, and threshold — Part 1

Input:

errors_timeseries.csv with columns: err_lstm, err_ae, Anomaly (optional).

Grids: $\mathcal{A}$ (alpha), $\mathcal{B}$ (beta), $\mathcal{K}$ (persistence), $\mathcal{P}$ (percentiles).

FPR bound: $f_{\max} \in (0, 1)$.

Output: Table with (α, β, K, t_h) and per-window metrics; global optimum (see Part 2).

```

1  1. Load and validate data
2  Read errors_timeseries.csv; check err_lstm, err_ae; Anomaly optional.
3  2. Robust standardization (median/MAD)
4   $(z_L, \text{med}_L, \text{MAD}_L) \leftarrow \text{RobustZ}(\text{err\_lstm});$ 
5   $(z_A, \text{med}_A, \text{MAD}_A) \leftarrow \text{RobustZ}(\text{err\_ae}).$ 
6  3. Initialize accumulators
7  Rows  $\leftarrow [];$ 
8  Best  $\leftarrow \text{None}.$ 
9  4. Sweep over  $K, \alpha, \beta$ 
10 foreach  $K \in \mathcal{K}$  do
11   foreach  $\alpha \in \mathcal{A}$  do
12    foreach  $\beta \in \mathcal{B}$  do
13     score  $\leftarrow \alpha z_L + \beta z_A;$ 
14      $\mathcal{T} \leftarrow \text{PercentileGrid}(\text{score}, \mathcal{P});$ 
15     BestABK  $\leftarrow \text{None}.$ 
16     5. Threshold sweep and persistence
17     foreach  $t_h \in \mathcal{T}$  do
18      raw  $\leftarrow \mathbb{K}\{\text{score} > t_h\};$ 
19       $\hat{y} \leftarrow \text{ApplyPersistence}(\text{raw}, K).$ 
20      if Anomaly available then
21       m  $\leftarrow \text{ConfusionMetrics}(y, \hat{y})$  // Precision, Recall,  $F_1$ , FPR,
          etc.
22       Append to Rows:  $(\alpha, \beta, K, t_h, m).$ 
23       Local selection under bound ( $\text{FPR} \leq f_{\max}$ ) with triple tie-break
24       if  $m.\text{FPR} \leq f_{\max}$  and  $(\text{BestABK} = \text{None} \text{ or } m.\text{recall} > \text{BestABK}.\text{recall}$ 
          or  $(m.\text{recall} = \text{BestABK}.\text{recall} \wedge m.F1 > \text{BestABK}.F1) \text{ or } (m.\text{recall} =$ 
           $\text{BestABK}.\text{recall} \wedge m.F1 = \text{BestABK}.F1 \wedge m.\text{FPR} < \text{BestABK}.\text{FPR})$ 
          then
25        BestABK  $\leftarrow (\alpha, \beta, K, t_h, m)$ 
26      end
27     end
28     else
29      Append to Rows:  $(\alpha, \beta, K, t_h, \text{NA})$  // no labels: suggest p99,
          e.g.
30     end
31   end
32   6. Local fallback if no candidate meets the bound
33   if Anomaly available and BestABK = None then
34    Among rows in Rows with  $(\alpha, \beta, K)$ , pick min FPR; if tie, max Recall;
35    assign to BestABK.
36   end
37 end
38 end
39 end

```

Algorithm 12: Offline tuning (continued)

```

40 7. Global selection and output persistence
41 7a. Update global optimum Best with the same triple tie-break:
42 max Recall ( $\text{FPR} \leq f_{\max}$ ); tie-break by  $F_1$ ; third tie-break by FPR.
43 7b. Write table  $(\alpha, \beta, K, t_h)$  with metrics and (med, MAD);
44 report  $(\alpha^*, \beta^*, K^*, t_h^*)$  and associated metrics.

45 Auxiliary functions
46 Func RobustZ ( $x[1:N]$ )
47    $m \leftarrow \text{median}(x)$ ;
48    $d \leftarrow \text{median}(|x - m|) + \varepsilon$ ;
49   return  $((x - m)/d, m, d)$ .
50 Func ApplyPersistence ( $b[1:N], k$ )
51   if  $k \leq 1$  then
52     return  $b$ 
53   end
54    $out \leftarrow \mathbf{0}, run \leftarrow 0$ 
55   for  $i \leftarrow 1$  to  $N$  do
56     if  $b[i] = 1$  then
57        $run \leftarrow run + 1$ ;  $out[i] \leftarrow \mathbb{I}\{run \geq k\}$ 
58     else
59        $run \leftarrow 0$ ;  $out[i] \leftarrow 0$ 
60     end
61   end
62   return  $out$ .
63 Func ConfusionMetrics ( $y[1:N], \hat{y}[1:N]$ )
64   Compute TP, FP, TN, FN; Precision, Recall,  $F_1$ , FPR;
65   return dictionary.
66 Func PercentileGrid ( $s[1:N], \mathcal{P}$ )
67   return  $\{ \text{Percentile}_p(s) : p \in \mathcal{P} \}$ .

```

B ARTIFACTS

Stage	File	Type	Use
AE	scaler.pkl	Scaler	Feature scaling for the AE.
AE	autoencoder_full.pth	Model (PyTorch)	Full reconstruction.
AE	encoder_full.pth	Model (PyTorch)	Latent embedding extraction.
AE	encoder.pth	TorchScript	Encoder deployment/inference.
AE	autoencoder_losses.png	Figure	Loss (MSE) per epoch.
AE	autoencoder_mae.png	Figure	MAE per epoch.
AE	robot_embeddings.csv; embeddings_train.npy; embeddings_val.npy	Data	Latent embeddings (train/val).
LSTM	scaler_lstm.pkl	Scaler	Latent scaling for the LSTM.
LSTM	lstm_predictor_state.pth	state_dict	Predictor weights.
LSTM	lstm_predictor_full.pth	Model (PyTorch)	Full module.
LSTM	lstm_predictor_ts.pt	TorchScript	Deployment/inference.
LSTM	lstm_pred_threshold.json	JSON	Operating thresholds ($\mu+3\sigma$, $p99$, $p99.7$).
LSTM	lstm_training_mse.png	Figure	Training MSE curve.
LSTM	lstm_training_metrics.csv	CSV	Per-epoch metrics (train/val).
LSTM	meta.json	JSON	Configuration (D , T , H , η).

Table 10: Artifacts generated by the AE+LSTM pipeline and their purpose.