國立陽明交通大學
電機資訊國際學位學程
碩士論文

EECS International Graduate Program
National Yang Ming Chiao Tung University
Master Thesis

區分攻擊與故障：應用於智慧物理系統的兩階段多源異常偵測方法

Differentiating Attacks from Faults: Two-stage Multi-datasource Anomaly Detection in Cyber Physical Systems

研 究 生：安伯托（Partho Adhikari）

指導教授：林盈達（Ying-Dar Lin）

中華民國一一四年六月

June 2025

區分攻擊與故障：應用於智慧物理系統的兩階段多源異常偵測方法

Differentiating Attacks from Faults: Two-stage Multi-datasource Anomaly Detection in Cyber Physical Systems

研究生：安伯托　　Student: Partho Adhikari

指導教授：林盈達博士　　Advisor: Ying-Dar Lin

國立陽明交通大學

電機資訊國際學位學程

碩士論文

A Thesis

Submitted to EECS International Graduate Program

College of Electrical Engineering and Computer Science

National Yang Ming Chiao Tung University

in partial Fulfilment of the Requirements

for the Degree of

Master in Science

in

Electrical Engineering and Computer Science

June 2025

Taiwan, Republic of China

中華民國一一四年六月

區分攻擊與故障：應用於智慧物理系統的兩階段多源異常偵測方法

學生：安伯托　　指導教授：林盈達 博士

國立陽明交通大學 電機資訊國際學位學程

摘　　要

智慧物理系統（Cyber-Physical Systems, CPS）正面臨日益多樣化的網路攻擊；這些攻擊在外部症狀上常與系統故障高度相似，使得異常分類格外困難。由於攻擊與故障的成因與因應策略截然不同，若無法精準區分，將嚴重影響系統可用性與安全性。本論文提出一種兩階段、多資料來源的異常偵測框架，以解決傳統單一來源或單階段方法之侷限。第一階段使用僅以正常資料訓練的單類分類器偵測行爲偏差；第二階段則在有限的標記資料下，將偵測到的異常區分爲「系統故障」或「網路攻擊」。爲補足單一來源資訊不足的問題，框架整合感測器數據、系統日誌與網路流量三種異質來源，並透過時間窗對齊與特徵集成，提高對跨層次威脅的可偵測性。實驗結果顯示，所提方法在公開 CPS 水瓶裝填模擬平台上可達到 0.99 的 F1 分數；相較於最佳單一資料來源的兩階段方法提升 15%，相較於單一資料來源單階段方法提升 27%，並較多資料來源單階段方法提升 7%。在所測試之模型中，單類支持向量機（OCSVM）在三種資料來源的兩個階段皆展現最優異的準確率。本研究驗證了多來源資訊與分階段學習策略在智慧物理系統異常偵測上的效益，並爲實務部署提供一套穩健且可擴充的解決方案。

關鍵詞：智慧物理系統、異常偵測、多來源集成、單類學習、兩階段方法、時間窗對齊

Differentiating Attacks from Faults: Two-stage Multi-datasource Anomaly Detection in Cyber Physical Systems

Student: Partho Adhikari Advisor: Dr. Ying-Dar Lin

EECS International Graduate Program
National Yang Ming Chiao Tung University

Abstract

Cyber-Physical Systems (CPS) face growing threats from cyber attacks that closely resemble system faults, making accurate anomaly classification particularly challenging. This problem is intensified in dynamic CPS environments, where the causes and responses of faults and attacks differ, yet their observable symptoms often overlap. Traditional single-stage, single-source data detection methods struggle to capture the full system context and lack robustness to novel threats due to limited labeled data and a narrow input scope. To address these limitations, this thesis proposes a two-stage, multi-datasource anomaly detection framework. The first stage uses one-class classifiers trained solely on normal data to detect behavioral deviations. The second stage classifies these anomalies as system faults or cyber attacks using only fault-labeled samples. To provide richer context, three heterogeneous data sources—sensor readings, system logs, and network traffic—are temporally aligned via time-windowing and processed independently before ensemble-based decision making. Experimental results show that the proposed framework achieves an F1-score of 0.99, representing improvements of 15% over the best single-datasource two-stage method, 27% over the single-datasource single-stage method, and 7% over the multi-datasource single-stage method. Among all tested models, One-Class SVM (OCSVM) emerged most effective in both stages for all datasources. The framework offers a robust, scalable solution for reliable anomaly detection in real-world CPS deployments.

Keywords: Cyber-Physical Systems, Anomaly Detection, Multi-source Alignment, One-Class Learning, Two-Stage Detection, Time-Windowing

Acknowledgement

Life flows swiftly like a stream, yet memories bloom endlessly in the passage of time.

Prof. Ying-Dar Lin, Prof. Yuan-Cheng Lai, and Prof. Ren-Hung Huang —I sincerely thank you for your invaluable guidance throughout this research journey. Your thoughtful advice during every discussion taught me to approach problems with logical rigor and careful attention to detail. You often reminded us to first ask ”Why,” before considering ”What” and ”How,” and you introduced the X—Y—Z strategy (X: solution, Y: problem, Z: system) as a framework for structuring research. These lessons have greatly shaped my academic thinking and will continue to benefit my future development.

Prof. Didik Sudayana —thank you for your guidance and support throughout these two years, including your careful review of my thesis. Your insights and feedback have been greatly appreciated and contributed meaningfully to this work.

I would also like to express my heartfelt thanks to all my labmates and everyone who has been part of this journey. Thank you for making these two years vibrant and colorful, and for your unwavering support in every situation—whether academic or personal. Your encouragement, collaboration, and kindness have made this experience both meaningful and memorable.

Lastly, I am deeply grateful to my friends and family for their constant support, patience, and understanding throughout this journey. Your presence has been my greatest strength.

Partho Adhikari

June 1, 2025

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cyber-Physical Systems (CPS), which tightly integrate computing, networking, and physical processes, are foundational to modern industrial and infrastructure systems. These systems inherently span both Information Technology (IT) and Operational Technology (OT) domains, where IT handles data and control logic, and OT includes both electrical and mechanical components responsible for real-world actions [1]. While this integration has enabled advanced automation and enhanced operational efficiency, it has also introduced complex and multifaceted challenges [2]. One of the most pressing challenges in this context is the reliable differentiation between cyber attacks and system faults—both of which can compromise system integrity and safety [3]. Under normal operating conditions, all components—digital, electrical, and mechanical—function in coordination according to predefined logic. Deviations from this expected behaviour can arise from system faults (e.g., hardware degradation or mechanical failure) or cyber attacks (e.g., data manipulation or command injection). While both lead to anomalous behaviour, their underlying causes, intentions, and required responses differ significantly. Because these anomalies often exhibit similar outward symptoms, determining whether they result from a fault or a cyber attack remains a considerable challenge.

This ambiguity presents a serious challenge. For example, an attacker may spoof a sensor reading to simulate overheating, while in reality, the temperature is normal. This can mimic the exact behavior of a degraded sensor or a stuck valve a typical fault condition. If such a case is misclassified as a fault, the adversary's activity may go unnoticed and escalate. Conversely, a true mechanical fault treated as an attack could lead to unnecessary shutdowns, incorrect remediation, or operational delays. Because CPS involve physical systems, these decisions carry real-world consequences, especially in safety-critical infrastructure such as energy, manufacturing, and transportation. The real-world consequences of these misclassifications are evident in high-impact incidents. For example, the 2015 cyberattack on Ukraine's power grid, which

caused widespread outages for over 225,000 users [4], illustrates the severe risks associated with inadequate anomaly diagnosis. Therefore, the needs of an effective anomaly detection must go beyond simple identification of abnormal behaviour; it must correctly distinguish system fault from attacks by leveraging all relevant data sources, as the evidence of abnormal behavior is distributed across multiple data sources.

Previous studies on anomaly detection in CPS address only part of the broader security challenge. Many works focus narrowly on intrusion detection using single source data, such as network traffic or system logs [5, 6], which may not capture the overlapping attack scenario with normal / faults. Others specialize in system fault detection through statistical analysis, yet fail to account for adversarial behavior that can mimic legitimate failures [7]. Although some recent research attempts to handle both faults and attacks [3], these approaches are often limited by simplified testbeds, lack of real fault scenarios, or the use of only one data source, typically sensor data [3], [8], [9], [10]. These limitations hinder the ability to accurately distinguish between cyber attacks and system faults, an essential requirement to implement effective and context-sensitive security solutions in real-world CPS environments.

Single-source detection models are limited in both accuracy and contextual understanding. For example, a model solely based on network traffic may incorrectly classify a sudden burst of packets as a cyber attack, while sensor data may show that the system is operating normally, leading to a false positive. Conversely, a sensor-based model might misinterpret a drifting temperature reading as an attack, even when logs and traffic patterns indicate no signs of malicious activity. These misclassifications highlight the importance of using multi-datasource approaches, which combine sensor readings, system logs, and network traffic to provide a more comprehensive view of system behavior. By incorporating multi-datasource, the system can cross-reference information, improving reliability and reducing both false positives and false negatives, thereby ensuring more accurate decision-making.

However, merely combining multiple data sources is insufficient. A more structured approach is required to tackle the ambiguity between faults and attacks. In real-world CPS environments,

normal operations are stable, whereas faults and attacks are more sporadic. This distinction allows for the easier collection of normal data, which is steady compared to rare and evolving attack data. Single-stage models struggle to handle rare attacks because they attempt to classify all anomalies at once, making it difficult to differentiate between faults and attacks. Our two-stage framework addresses this by first detecting anomalies using only normal data, and then classifying the detected anomalies as faults or attacks in second stage, which is trained on fault data. This modular approach improves precision and reduces misclassification of rare attacks.

Our work makes three key contributions to anomaly detection in CPS, addressing gaps identified in existing studies:

1. We address the problem in OT cyber-physical anomaly detection by developing a method to accurately distinguish between cyber attacks and system faults—two distinct anomaly classes that often exhibit overlapping observable behaviors but differ fundamentally in their underlying causes, temporal patterns, and required mitigation strategies.

2. We propose a two-stage multi-datasource detection framework that integrates network traffic, system logs, and sensor data, leveraging unsupervised learning in first stage to detect deviations from normal behavior, followed by supervised classification in second stage to differentiate attacks from faults, thereby enhancing contextual analysis and enabling more precise and actionable system responses.

3. We validate the proposed solution effectiveness by evaluating single-datasource versus multi-datasource combinations and single-stage versus two-stage detection approaches, demonstrating improved detection accuracy and resilience against unseen threats.

The remainder of this thesis is structured as follows: Chapter 2 provides a comprehensive review of the background and related work, laying the foundation for our study. Chapters 3 and 4 articulate the problem formulation and propose our solutions, respectively. Chapter 5 delineates the implementation details of our framework, while Chapter 6 presents and analyzes the experimental results. Finally, Chapter 7 offers concluding remarks and outlines future research directions.

Chapter 2

Background and Related Work

This chapter provides key technical background for the proposed anomaly detection framework. It covers the structure and behavior of CPS, the nature of cyber attacks and system faults in OT, and the role of machine learning in OT anomaly detection. We conclude with a review of related studies to position our work within the broader research landscape.

2.1 CPS

In CPS, embedded controllers and networking infrastructure interact with sensors and actuators to monitor and influence real-world operations. These systems are foundational to industrial and infrastructure domains such as manufacturing, energy distribution, and water treatment [3].

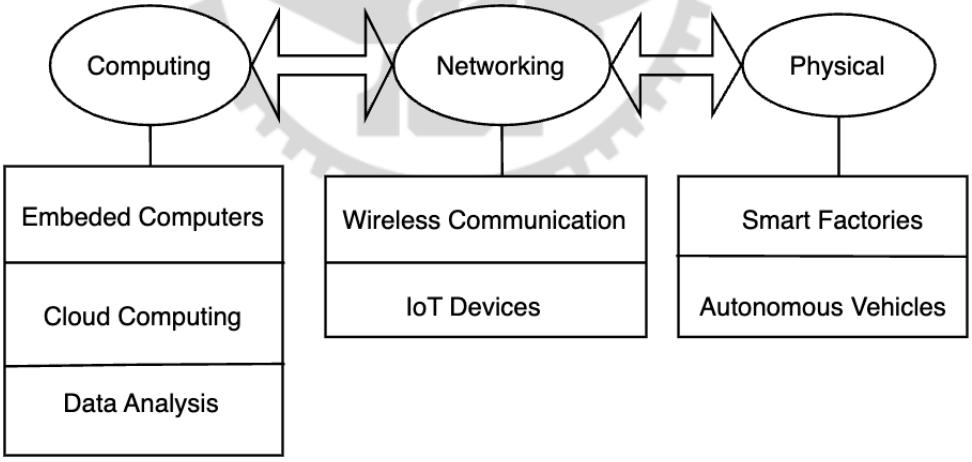


Figure 2.1: CPS Architecture: Computing, Networking, and Physical Layers

As illustrated in Figure 2.1, CPS are typically organized into three layers: computing, networking, and physical. The computing layer encompasses devices like PLCs and edge/cloud platforms that execute control logic and data analytics, including AI-driven decision-making [11]. The networking layer ensures timely data exchange across components using industrial

protocols and IoT connectivity [10]. The physical layer consists of sensors, actuators, and mechanical systems that perform or measure actions in the physical environment [12]. These layers operate in a feedback loop for instance, in a water bottling system, sensor data informs control actions that regulate pumps and valves to maintain operational flow.

A CPS emulator is a simulation platform designed to replicate the behavior of real-world CPS environments by integrating physical processes with computing and networking components. These emulators are essential for generating data to develop and test anomaly detection systems, providing a controlled environment to simulate both normal and anomalous system behaviors. However, many available CPS emulators have limitations that prevent them from fully capturing the complexity of real-world CPS environments.

Existing CPS emulators, such as SWaT [3], WADI [3], and ICSSIM [13], often focus on narrow aspects of CPS, such as sensor data or network-based attacks. While these systems provide valuable insights, they do not offer a complete representation of cyber-physical interactions. For instance, SWaT and WADI mainly simulate sensor-level data, lacking the integration of system logs or network traffic, which restricts their ability to evaluate multi-datasource anomaly detection approaches. Additionally, some emulators are confined to network-level simulations and lack realistic fault injection or physical process feedback mechanisms. These shortcomings make it challenging to generate diverse, multi-datasource data for training anomaly detection models.

The CPS Water Filling Emulator [13] is a widely used simulation platform designed to emulate industrial control systems in a water bottling process. It features a modular architecture with two independently functioning PLCs, which control sensor-actuator dynamics across different stages of the bottle filling system. This dual-PLC configuration enables detailed simulation of physical and cyber interactions in CPS environments. Additionally, the emulator is containerized using Docker, which simplifies deployment and enhances scalability. Its flexible design supports the integration of new components, making it suitable for generating diverse datasets, including synchronized sensor data, system logs, and network traffic.

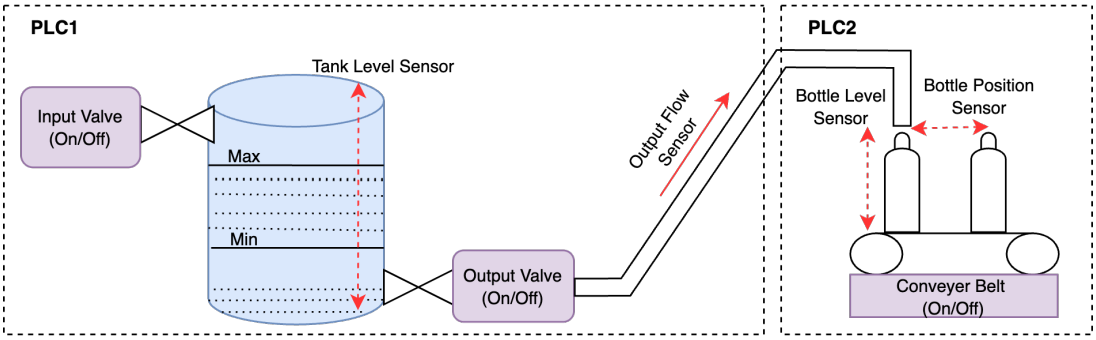


Figure 2.2: CPS Water Filling Emulator

A key advantage of this emulator is its ability to produce synchronized multi-datasource, which enables the evaluation of anomaly detection models across the physical, networking, and computing layers of CPS. Among these sources, log data plays a particularly critical role—despite its slower detection time and lower standalone accuracy. In real-world industrial systems, logs generated by PLCs, SCADA interfaces, and control scripts provide semantic information that is often absent from raw sensor or network data. For example, in a water bottling scenario, a valve remaining open longer than expected could result from a physical blockage (observable in sensor readings) or a misconfigured control command (captured in logs). While sensor data reflects the outcome (e.g., abnormal tank level) and traffic data may show communication irregularities, logs can reveal system-level events such as configuration changes or command overrides that explain anomalous behavior. This makes logs a valuable complementary signal in multi-datasource detection, especially for identifying cyber-induced faults that may not manifest directly in physical measurements. Therefore, although log data alone may yield lower F1-scores due to its sparsity and noise, its inclusion improves detection coverage in scenarios where cyber actions precede or influence physical effects.

2.2 Attacks: IT vs. OT

In CPS, cyber attacks frequently originate in the IT network and then propagate into the OT layer, exploiting the lack of integrated visibility and tailored detection mechanisms. Figure 2.3 illustrates a representative IT-to-OT attack chain, highlighting how initial misclassification in the IT layer can lead to physical damage in the OT environment.

The attack begins with a phishing email sent to an IT endpoint. This results in a macro-based execution that triggers an alert on the Endpoint Detection and Response (EDR) system. The alert is forwarded to a Security Information and Event Management (SIEM) system and escalated to the Security Operations Center (SOC). However, it is misclassified as a false positive, leading to a containment action on a clean host—diverting resources while missing the actual threat.

The adversary then deploys a second-stage malware that bypasses detection and spreads laterally into the OT network. Once inside the OT environment, it initiates malicious actions such as injecting abnormal control commands into Programmable Logic Controllers (PLCs). These are eventually flagged by an OT-specific Intrusion Detection System (IDS), but the detection is delayed. The SIEM system finally recognizes the intrusion, prompting the SOC to escalate the incident to the OT response team. However, by then, critical systems may already be compromised—resulting in actuator disruption and PLC failure.

Such attacks can be launched using a wide range of penetration testing tools and Python-based libraries. Tools like `Metasploit` is commonly used to exploit IT-side vulnerabilities. On the OT side, frameworks such as `ICSploit`, `Scapy`, `PLCSIM`, and even legitimate industrial software like `STEP 7` (used for programming Siemens PLCs) can be misused to craft malicious packets, manipulate control logic, or spoof sensor values. These tools support the exploitation of industrial protocols such as Modbus, DNP3, and S7, enabling adversaries to disrupt physical processes with precision.

This scenario underscores the fundamental limitation of IT-centric security approaches: they are not designed to understand or protect physical-layer behaviors. Without an OT-specific anomaly detection strategy tailored to the unique dynamics of CPS environments, organizations are exposed to delayed detection, incorrect classification, and potentially irreversible physical damage.

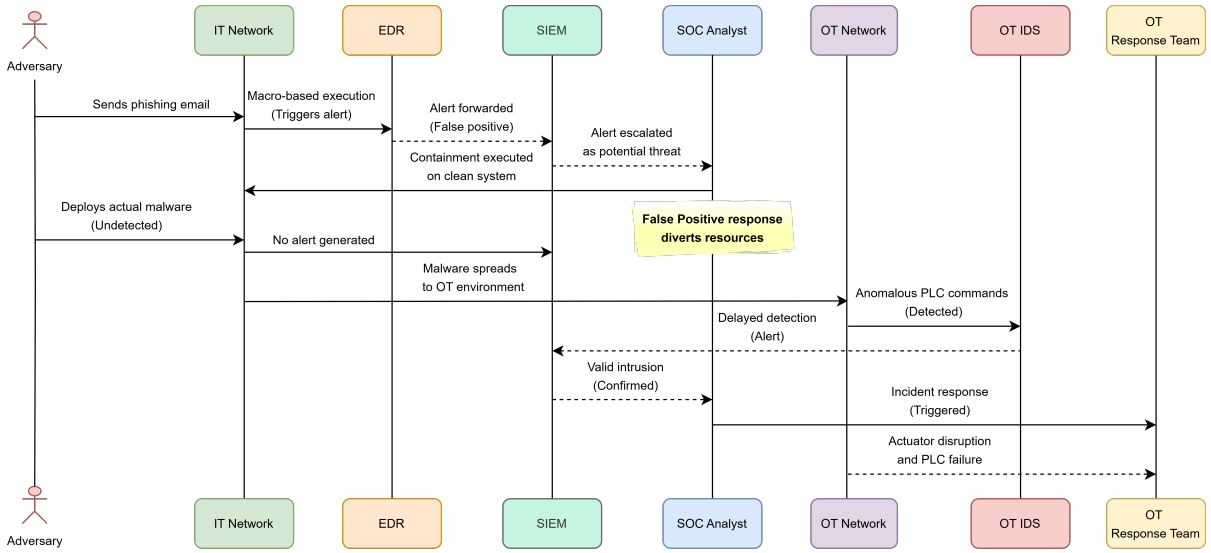


Figure 2.3: Illustration of an end-to-end attack chain crossing from IT into OT networks, highlighting gaps in detection and delayed response across layers.

2.3 System Faults in OT

System faults in OT environments originate from physical degradation, environmental interference, or misconfigurations—rather than adversarial intent. Unlike cyber attacks, which are deliberately induced, system faults occur naturally or due to human error, but they often produce similar symptoms at the system level.

In the CPS system faults can be generated by deliberately modifying PLC configuration files or introducing physical process delays. For example, adjusting the valve timing parameters in PLC2 can simulate actuator sticking, while altering sensor calibration in PLC1 can emulate sensor drift. These faults affect the closed-loop interaction between PLCs, sensors, and actuators —disrupting normal system operations without triggering immediate security alerts.

Typical faults include:

- **Sensor drift**: Gradual deviation in sensor output due to aging components or environmental noise. This may cause PLCs to make incorrect control decisions.
- **Actuator sticking**: An actuator (for example, valve or motor) fails to respond promptly, halting processes such as water filling or bottle transfer.

- **Memory corruption**: Faulty memory in embedded systems can lead to misinterpretation of inputs, such as incorrect tank level readings.
- **Overheating**: Environmental stress on hardware components can degrade performance or cause shutdowns, especially in high-load CPS like power grids.

Although system faults are non-malicious in nature, their impact on process safety and availability can be just as severe as cyber attacks. Moreover, their physical manifestations often resemble those of adversarial behavior, making it essential to distinguish between the two using advanced detection techniques.

2.4 Machine Learning based Anomaly Detection in OT

Traditional rule-based and signature-based detection methods, though effective for known threats, fall short in identifying zero-day attacks and novel system faults due to their dependence on predefined patterns [5]. Machine Learning (ML) provides a robust alternative by employing data-driven models to detect anomalies, including previously unseen ones, through the identification of intrinsic patterns within operational data. ML approaches are broadly classified into two categories: Supervised Learning (SL) and Unsupervised Learning (UL), each offering distinct detection mechanisms and data requirements that enhance their applicability across diverse OT environments.

Figure 2.4 exemplifies this distinction by illustrating decision boundaries for One-Class Support Vector Machine (OCSVM) and Support Vector Machine (SVM). In the left plot, OCSVM, an unsupervised technique, is trained solely on normal data (represented as blue points), defining a boundary that encircles this data; anomalous instances (red crosses) outside this boundary are identified as outliers, enabling detection without prior anomaly knowledge. In contrast, the right plot depicts SVM, a supervised method, trained on labeled datasets with both normal (blue) and anomalous (red) classes, constructing a classification boundary to differentiate between them, which requires labeled anomaly data for optimal performance [14], [15]. This adaptability, particularly the ability of UL to handle unlabeled data, underscores ML's critical role in bolstering

the resilience of OT systems against evolving threats, making it an indispensable tool for modern cybersecurity frameworks.

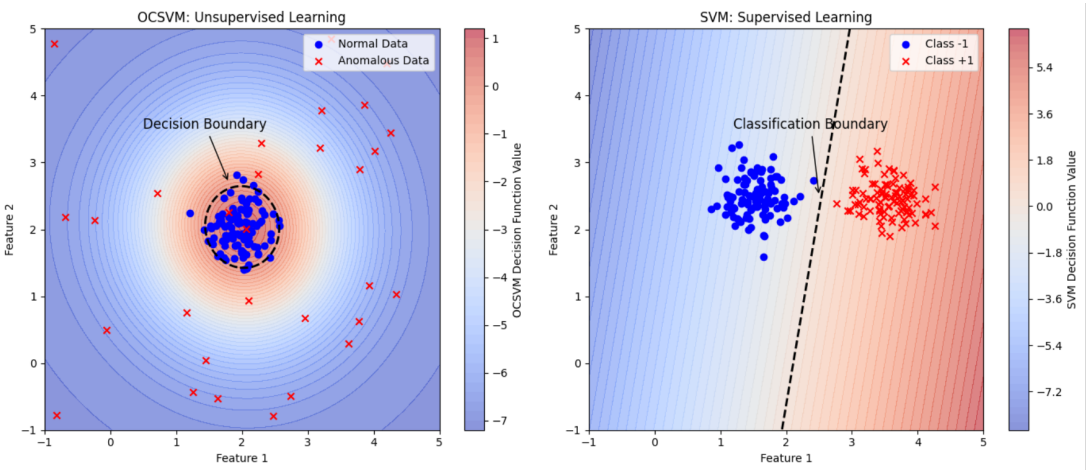


Figure 2.4: Comparison of UL vs. SL Decision Boundaries

2.5 Related Work

To better position our work within the existing landscape, we conducted a structured analysis of recent anomaly detection methods in CPS. Table 2.1 presents a comprehensive evaluation of prior research on anomaly detection within CPS, assessing key dimensions such as target anomaly types, data sources, detection models, detection stages, and dataset usage. This analysis reveals significant gaps in existing methodologies, particularly in addressing the complex interplay between cyber-attacks and system faults, which our proposed multi-datasource, ensemble-based, two-stage detection framework aims to overcome through improved contextual integration and classification accuracy.

Anomaly detection in CPS has been widely studied, with prior approaches often constrained by their data sourcing strategies. Single-datasource approaches—relying on isolated streams such as network traffic [17], system logs [20], or sensor data [21]—often suffer from limited contextual understanding, leading to elevated false alarm rates. In contrast, leveraging multiple data sources allows systems to capture diverse indicators of anomalous behavior and enables detection from multiple perspectives; if one source fails to reveal an anomaly, others may still

Table 2.1: Comparison of Related Work in CPS Anomaly Detection

Paper	Targets	Data Sources			Detection Method			Detection Stage	Dataset
		T	L	S	SL	UL	Other		
[16]	Normal/Attack	-	-	O	LSTM, GCN, GAN	-	Digital Twin	1	SWaT, WADI, Batadal
[17]		O	-	O	Siamese CNN	-	-	1	UNSW-NB15
[18]		O	-	O	LR, NB, SVM, KNN, MLP	-	-	1	Edge-IIoTset2023, CICIoT2023
[19]		O	-	O	GAN	LSTM-Autoencoder	-	1	Water Distribution
[20]		O	-	O	-	OC-KNN, OCSVM	-	1	Self-generated
[11]		O	-	-	LSTM	-	SARIMA	2	Self-generated
[21]	Normal/Fault	-	-	O	-	-	Property Mining	1	Self-generated
[22]		-	-	O	-	-	Mathematical Modeling	1	Self-generated
[23]		-	-	O	LSTM, Temporal Convolution	-	-	1	Self-generated
[24]	Attack/Fault	-	-	O	-	-	Mathematcial Modeling	1	Self-generated
[25]		-	-	O	-	-	Mathematcial Modeling	3	Self-generated
[12]		-	-	O	-	-	Rule-based (Fault)	2	Self-generated
Ours		O	O	O	OCSVM	OCSVM	-	2	Self-generated

provide critical signals for accurate identification. While some multi-datasource methods attempt to integrate diverse data streams [12], they typically rely on joint input features without source-specific reasoning or modular decision logic, particularly when anomalies manifest subtly across multiple layers of the system. As a result, these models struggle to generalize when one modality is noisy, missing, or adversarially manipulated, limiting their ability to reduce false negatives. Our framework addresses these shortcomings by combining sensor readings, system logs, and network traffic in an ensemble-based mechanism, analyzing each source independently and flagging an anomaly if any source detects one, thereby enhancing detection reliability.

The choice of detection models further shapes CPS anomaly detection outcomes. Deep learning techniques, such as Long Short-Term Memory (LSTM), Graph Convolutional Networks (GCN), and Generative Adversarial Networks (GAN) [16, 19, 11, 23], excel in capturing complex patterns but demand large labeled datasets, which are scarce in OT environments. For example, CURNet leverages recursive LSTM and temporal convolution to model temporal dependencies [23], while few-shot Siamese CNN tackles data scarcity but lacks interpretability [17]. Hybrid ensemble methods, combining Logistic Regression (LR), Naive Bayes (NB), SVM, KNN, and MLP [18], offer robustness but are computationally intensive. Mathematical and rule-based approaches, like Daikon for formal fault detection [21] and Inductive Rule-

Based Reasoning (Induct RDR) [26], provide precision but lack adaptability to evolving threats. Unsupervised anomaly detection techniques—such as OC-KNN, OCSVM [20], and LSTM-Autoencoders combined with GANs [19]—are often used in CPS settings due to the scarcity of labeled attack data. These methods learn patterns of normal behavior and identify deviations as anomalies without relying on labeled training sets. Our approach adopts an unsupervised OCSVM in the first stage to detect anomalies from multiple independent data sources. In the second stage, we incorporate supervised classification using only fault data, improving both the interpretability and precision of anomaly categorization in environments where attack labels are limited or incomplete.

The architecture of an anomaly detection pipeline plays a pivotal role in securing CPS. Many existing approaches adopt a single-stage detection framework, wherein multi-class classifiers are trained directly on labeled attack data to detect and categorize anomalies. However, this design introduces two critical limitations. First, overreliance on known attack data reduces the model's robustness against unseen or evolving attack patterns, which is a common scenario in real-world CPS environments. As attack surfaces change and adversarial strategies grow more sophisticated, models trained only on historical attack signatures often fail to generalize. For instance, deep learning-based single-stage classifiers such as those used by Xu et al. [16] and Zhou et al. [17] offer strong performance on benchmark datasets, but degrade under novel threat scenarios. Second, mathematical modeling approaches, while interpretable and lightweight, often fall short in capturing the non-linear dynamics and operational diversity inherent in CPS. Works such as those by Zhang et al. [24] and Taheri et al. [25] use analytical models to jointly detect cyberattacks and physical faults, but such approaches require detailed domain-specific knowledge and assume idealized system behavior. This makes them brittle in practice and difficult to scale or adapt to heterogeneous systems and unexpected operational contexts. To overcome these limitations, this thesis proposes a two-stage, multi-datasource anomaly detection framework. In the first stage, an unsupervised OCSVM ensemble processes multiple data sources to identify anomalous behavior. In the second stage, a supervised classification model trained on labeled fault data to separate each fault samples from attacks. This layered design improves both detection precision and interpretability, facilitating more targeted and effective mitigation

strategies in CPS environments.

Dataset selection plays a pivotal role in evaluating the performance and generalizability of anomaly detection models in CPS. Widely used public datasets such as SWaT [3], WADI [3], UNSW-NB15 [5], CICIoT2023 [12], and EdgeIIoTset2023 [20] offer baseline environments and predefined attacks. However, these datasets have notable limitations. Most either focus solely on cyberattacks or system faults—rarely both—and often rely on single-source data, typically limited to sensor streams or network traffic. Additionally, some datasets, like UNSW-NB15 and CICIoT2023, are primarily tailored to IT network environments and may not reflect the specific operational dynamics of OT system. To overcome these limitations, our work introduces a self-generated dataset collected from the Waterfilling Testbed—a custom-built CPS emulator designed to reflect realistic industrial control environments. Unlike existing datasets, our dataset integrates three synchronized data modalities: sensor readings, system logs, and network traffic, each captured under three distinct operating conditions: normal, cyber attack, and system fault scenarios. This comprehensive coverage enables accurate detection across diverse anomaly types, supports fine-grained correlation across data layers, and facilitates evaluation of multi-datasource detection frameworks in time-sensitive environments.

Our framework improves upon existing CPS anomaly detection methods by integrating multi-datasource (traffic, logs, and sensors) and employing a two-stage detection pipeline. Unlike previous studies relying on single-datasource, our approach uses ensemble decision-making to reduce false negatives. Additionally, while many methods lack fault-attack distinction, our system first detects anomalies and then classifies them as attacks or faults, improving accuracy. We combine unsupervised learning in the first stage with supervised classification in the second to handle labeled attack data scarcity. Finally, Instead of relying on limited public datasets that lack synchronized multi-datasource and realistic fault or attack scenarios, we utilize a custom Water Filling Testbed to enable robust and practical evaluation.

Chapter 3

Problem Statement

This chapter introduces the notations used throughout the thesis and presents the core research problem: accurately distinguishing cyber attacks from system faults in CPS.

3.1 Notation Table

To ensure clarity and consistency throughout the thesis, we introduce a unified notation system that formalizes the key elements of the dataset and learning framework. Table 3.1 defines these notations, categorized into dataset and machine learning components.

In the dataset category, each instance x consists of multi-source features—sensor readings (x^S), system logs (x^L), and network traffic (x^T). Labels are denoted by $y \in \{0, 1, 2, 3\}$, where 0 is normal, 1 is anomaly, 2 is fault, and 3 is attack. Training and testing datasets are represented as D_w^R and D_w^P, respectively, with $w \in \{0, 1, 2\}$ indicating sensors, logs, and traffic.

In the machine learning category, ML_n denotes the set of candidate algorithms applied to each training set D_w^R from source w. For each source and detection stage $s \in 1, 2$, a classifier $M_w^{(s)}$ is trained. The optimal classifier for each source and stage, denoted as $M_w^{(s)*}$, is selected based on the highest F1-score. Predictions from these optimal classifiers are aligned by timestamp and aggregated using an ensemble decision method M^E, and the best-performing combination M_{Best}^E is selected for final decision-making.

3.2 Problem Overview

In CPS environments, both system faults and cyber attacks can produce similar effects, such as erroneous sensor readings, actuation delays, or control anomalies, yet their causes and implications differ significantly. Faults arise from physical or operational degradation, while cyber

Table 3.1: Notation Table

Dataset		
Dataset	D	$D = D_w^R \cup D_w^P$
Training Set	D_w^R	$D_w^R, w = 0, 1, 2$; 0: sensors data, 1: system log, 2: traffic flow
Training Sensors	D_0^R	$D_0^R = \{(x_i^S, y_a) \mid i \in [1, RS]\}$; RS: size of training sensors data
Training Log	D_1^R	$D_1^R = \{(x_j^L, y_a) \mid j \in [1, RL]\}$; RL: size of training log data
Training Traffic	D_2^R	$D_2^R = \{(x_k^T, y_a) \mid k \in [1, RT]\}$; RT: size of training traffic flow
Testing Set	D_w^P	$D_w^P, w = 0, 1, 2$
Testing Sensors	D_0^P	$D_0^P = \{(x_i^S, y_a) \mid i \in [1, PS]\}$; PS: size of testing sensors data
Testing Log	D_1^P	$D_1^P = \{(x_j^L, y_a) \mid j \in [1, PL]\}$; PL: size of testing log data
Testing Traffic	D_2^P	$D_2^P = \{(x_k^T, y_a) \mid k \in [1, PT]\}$; PT: size of testing traffic flow
Input	x	$x = \{x_b\}$; $x_b = (x_i^S, x_j^L, x_k^T)$; $x_i^S \in R^{NS}, x_j^L \in R^{NL}, x_k^T \in R^{NT}$; NS, NL, NT: size of sensors, log, traffic features respectively
Label	y_a	$y_a, a = 0, 1, 2, 3$; 0: normal, 1: anomalies, 2: faults, 3: attacks
Machine Learning		
Classifier	$M_w^{(s)}$	Classifier for data source w at stage $s \in \{1, 2\}$
Optimal Classifier	$M_w^{(s)*}$	$M_w^{(s)}$ with highest F1 score
Ensemble Decision	M^E	$M^E \in M$, where $M = \bigcup_w M_w^{(s)}$; Decision computed by timestamp-aligning outputs of optimal classifiers
Best Ensemble Decision	M_{Best}^E	M^E with highest F1 score
ML Model	M_n	$M_n = ML_n(D_w^R)$
ML Algorithms	ML_n	$0 \leq n < \lvert ML \rvert$, $\lvert ML \rvert$: number of ML algorithms

attacks result from intentional malicious actions such as spoofing, logic tampering, or command injection.

Conventional detection systems often rely on single-source inputs and a one-stage classification pipeline. These approaches are limited in scope and often inaccurate in practice. Specific challenges include:

- Misclassifying faults as attacks, leading to unnecessary shutdowns or alarms.
- Failing to detect attacks that mimic fault patterns.
- Relying on a single data source, creating a single point of failure and limiting context.

The detection task is further complicated by the nature of CPS cyber attacks:

- Attacks typically impact multiple subsystems, and their effects are not fully observable through any single modality.

- Attack patterns evolve, and labeled attack data is scarce and quickly becomes outdated, limiting the utility of supervised approaches.

These limitations highlight the need for a multi-datasource, modular detection strategy that can isolate anomalies. Our proposed solution adopts this principle by combining multiple data sources and structuring the detection in two stages.

3.3 Problem Formulation

CPS generate complex, heterogeneous data from sources such as sensor readings, control logs, and network traffic. These systems are vulnerable to both cyber attacks and system faults, which can produce similar symptoms—such as unexpected sensor values, delayed actuation, or erratic control behavior—despite having fundamentally different causes and requiring different responses.

The core problem lies in accurately distinguishing between cyber attacks and system faults based on multi-source observations. This is complicated by several challenges: (1) anomalies often exhibit overlapping patterns across sources, making their origin ambiguous; (2) attack data is scarce, difficult to annotate, and rapidly evolving in nature; and (3) single-source or monolithic detection models tend to suffer from false positives and misclassification due to limited contextual visibility. Therefore, the detection task must account for limited labeled data, temporal variability in attack behaviors, and the need for cross-source correlation to achieve reliable and interpretable anomaly diagnosis.

The problem addressed in this thesis can be formally defined as follows:

- **Input:**
  - D_w^R: Training dataset for data source $w \in \{0, 1, 2\}$, representing sensor, log, and traffic data respectively.
  - D_w^P: Corresponding test dataset for each source.

- ML_n: Set of candidate machine learning algorithms.

- **Output:**

  - $M_w^{(s)*}$: Optimal classifier for each data source w and stage s, selected based on the highest F1-score.

  - M_{Best}^E: Best ensemble decision across data sources, selected based on overall F1-score.

- **Objective:**

  - Maximize the F1-score of both stagewise individual classifiers $M_w^{(s)*}$ and the ensemble decision M_{Best}^E across test datasets.

Chapter 4

Two-stage Multi-datasource Approach

This chapter presents our complete anomaly detection solution for CPS, we first provide an overview of the complete solution and then describe each phase of the solution: training classifiers from individual data sources, performing two-stage detection, and selecting the best data-source combination through timestamp-aligned ensemble decision.

4.1 Solution Overview

This section presents the complete design of our anomaly detection framework for CPS, structured around a two-stage architecture. We begin by mapping key challenges to their corresponding solutions, followed by a step-by-step description of each system component. Figure 4.1 illustrates how each design decision addresses a specific problem in CPS anomaly detection.

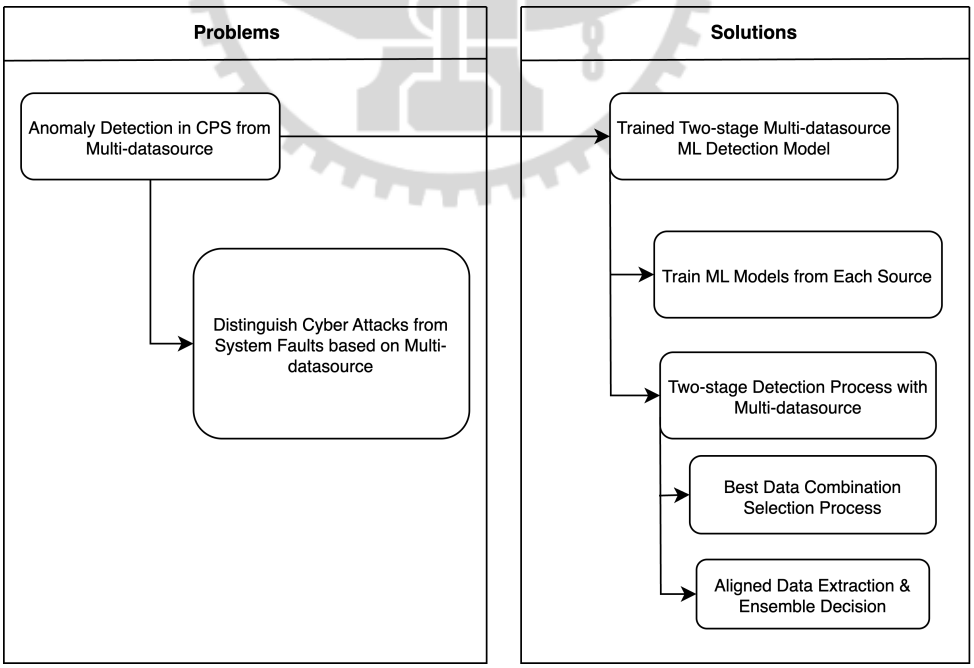


Figure 4.1: Mapping between identified problems and corresponding solutions

The first major challenge achieving accurate anomaly detection in CPS using multi-datasource inputs, is addressed by integrating sensor data, system logs, and network traffic into a unified detection pipeline. Traditional single-source models lack system-wide observability, whereas our approach leverages complementary signals from diverse data types to capture cross-domain anomalies.

The second challenge involves distinguishing cyber attacks from system faults. Although both manifest as abnormal behavior, they differ significantly in root causes and remediation strategies. Our architecture separates detection from diagnosis: the first stage ($M_w^{(1)}$) detects deviations from normal behavior using one-class classifiers trained on normal-only data (y_0), while the second stage ($M_w^{(2)}$) classifies detected anomalies as either faults (y_2) or attacks (y_3) using supervised models trained on labeled fault data.

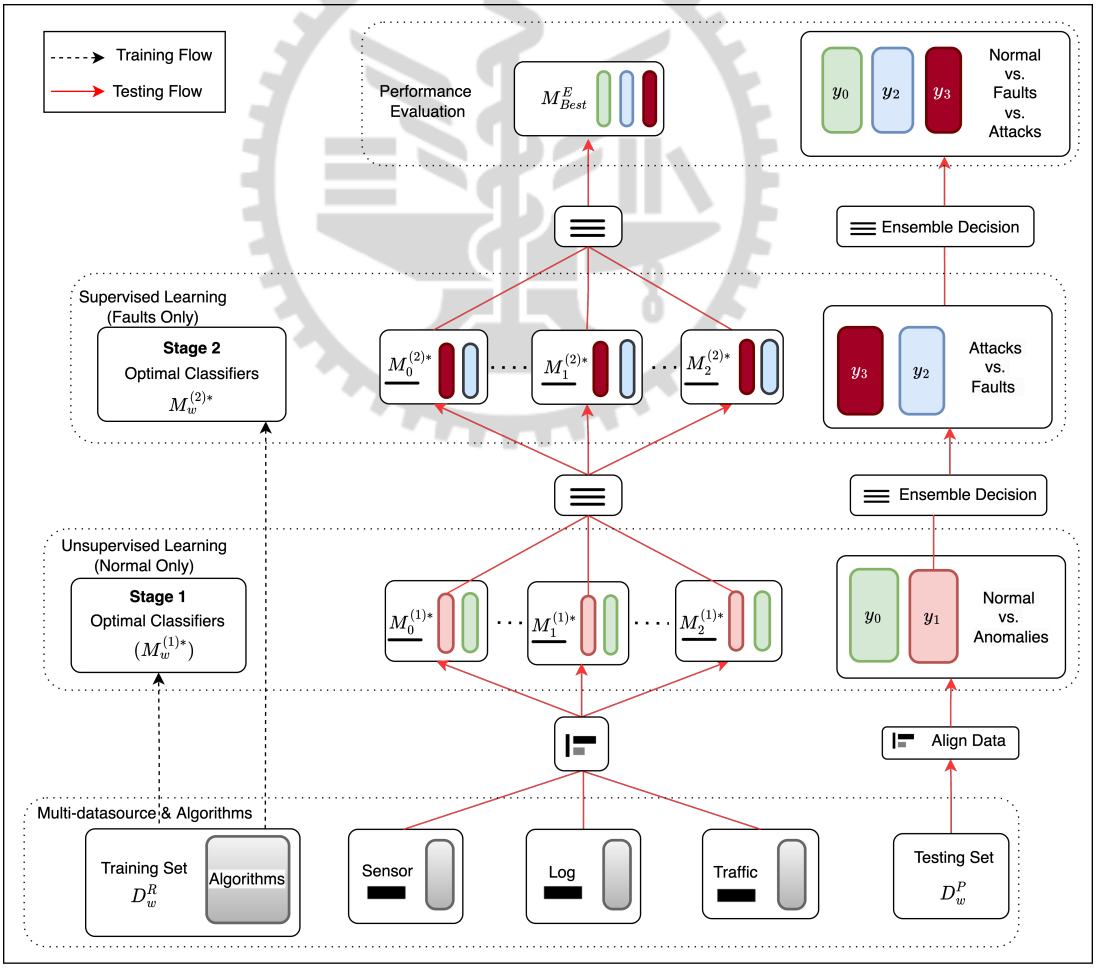


Figure 4.2: Complete Solution - Two-stage Multi-datasource Method

To consolidate the individual components into a cohesive system, we present the full architecture of our proposed framework. Figure 4.2 provides an end-to-end view of the framework, covering model training, classifier selection, anomaly detection, and final decision-making via time-aligned ensemble logic. This modular design supports robust, scalable deployment in real-world CPS settings, with built-in flexibility for system-specific adaptation and partial observability.

In the first stage, each test input is independently assessed by pretrained source-specific classifiers $M_w^{(1)*}$, trained exclusively on normal data. Each classifier evaluates whether the input deviates from expected patterns (i.e., outputs an anomaly label y_1). If all classifiers indicate normal behavior, the input is accepted; otherwise, it proceeds to the second stage for further analysis.

To manage CPS data heterogeneity, we train distinct machine learning models for each data source $w \in 0, 1, 2$ (sensor, log, traffic), each capturing different temporal or semantic features. For both stages, the optimal classifier $M_w^{(s)*}$ is selected from a candidate set ML_n, based on the highest validation F1-score.

To unify decisions across sources, we apply a timestamp-aligned ensemble mechanism. All data streams are synchronized using a fixed time window. During inference, predictions from the selected source-specific models are aggregated into a unified ensemble decision M^E. The best-performing source combination, denoted M_{Best}^E, is selected to maximize overall F1-score and ensure decision reliability.

4.2 Train ML Models from Each Datasource

This component is responsible for training and selecting the best-performing classifiers from each data source across both detection stages. Each source—sensor, log, and traffic, is handled independently to ensure that the unique characteristics of each modality are preserved during training.

The process begins by applying a set of ML algorithms ML_n to each data source $w \in \{0, 1, 2\}$ (representing sensor, log, and traffic data, respectively). Models are trained on the corresponding training dataset D_w^R and evaluated on D_w^P based on their F1-score. The model with the highest score is selected as the optimal classifier $M_w^{(s)*}$ for stage s.

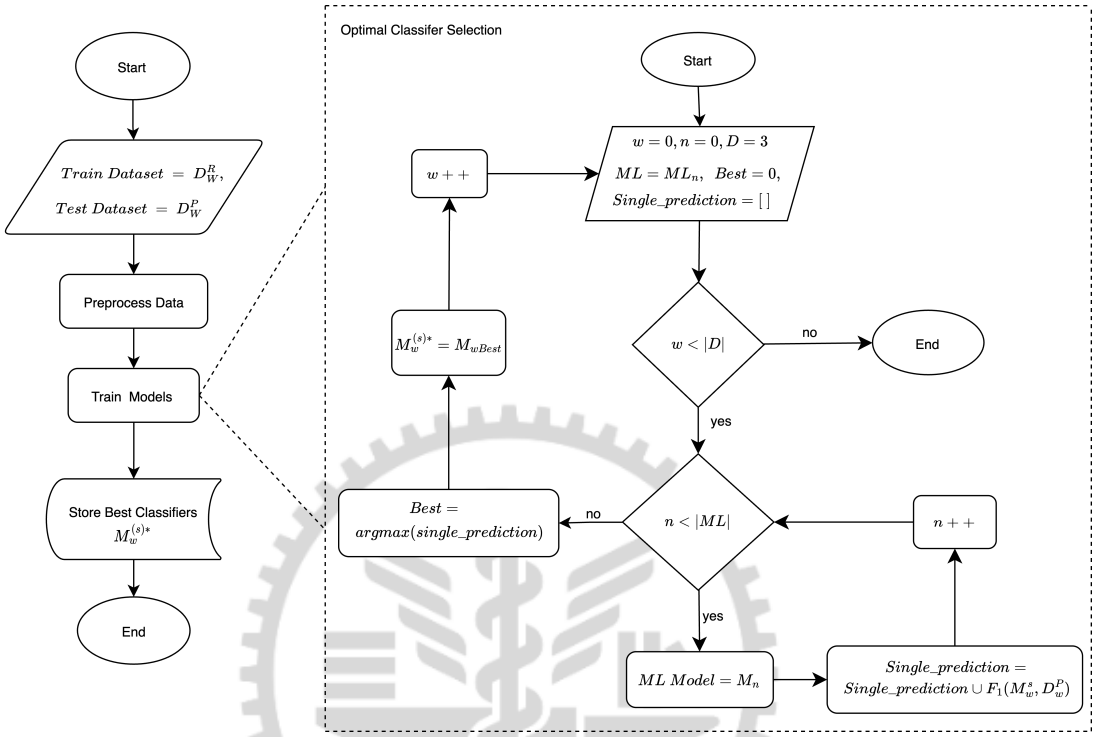


Figure 4.3: Workflow for Selecting Optimal Classifiers from Each Source

To ensure each data source is paired with the most effective model, we implement a structured classifier selection process. Figure 4.3 illustrates the step-by-step workflow of this selection process. For each data source, the pipeline iteratively tests different ML algorithms, stores their predictions, and selects the one achieving the highest F1 performance. This ensures that classifiers passed to the two-stage detection module are individually optimized for their respective sources and stages.

Formally, the process follows this logic:

- For each data source $w \in \{0, 1, 2\}$:
  - For each algorithm ML_n, train a model on D_w^R.
  - Evaluate on D_w^P and record F1-score.

– Select the model $M_w^{(s)*}$ with the highest score.

The selected classifiers $M_0^{(s)*}$, $M_1^{(s)*}$, $M_2^{(s)*}$ are stored for use in the two-stage detection pipeline and timestamp-aligned ensemble decision-making.

4.3 Two-Stage Detection Process with Multi-datasource

To clarify how the system detects and classifies anomalies using multi-datasource inputs, this section details the two-stage detection pipeline. As shown earlier in Figure 4.2, the proposed framework identifies behavioral anomalies and distinguishes cyber attacks from system faults using a modular, multi-datasource architecture.

The framework follows a structured, two-stage detection process designed for interpretability and robustness. In the first stage, each test input is independently assessed by pretrained, source-specific classifiers $M_w^{(1)*}$, trained exclusively on normal data (y_0). These models determine whether the behavior deviates from normal operation (i.e., predict an anomaly label y_1). If all classifiers agree the input is normal, it is accepted; otherwise, it proceeds to the second stage for further evaluation. In the second stage, classifiers $M_w^{(2)*}$—trained specifically on fault-only data (y_2)—assess whether the anomalous input reflects a benign system fault or a malicious cyber attack (y_3). The outputs of the individual classifiers are then aggregated using an ensemble approach to form the final decision.

To improve resilience, the system uses an OR-rule ensemble strategy: if any source flags an input as anomalous, the corresponding time step is labeled accordingly. This reduces false negatives and supports partial observability by treating missing inputs from any source as neutral in the final decision.

The following subsection discusses how to select the most effective combination of data sources and align their outputs with the ensemble decision process to maximize detection accuracy and robustness.

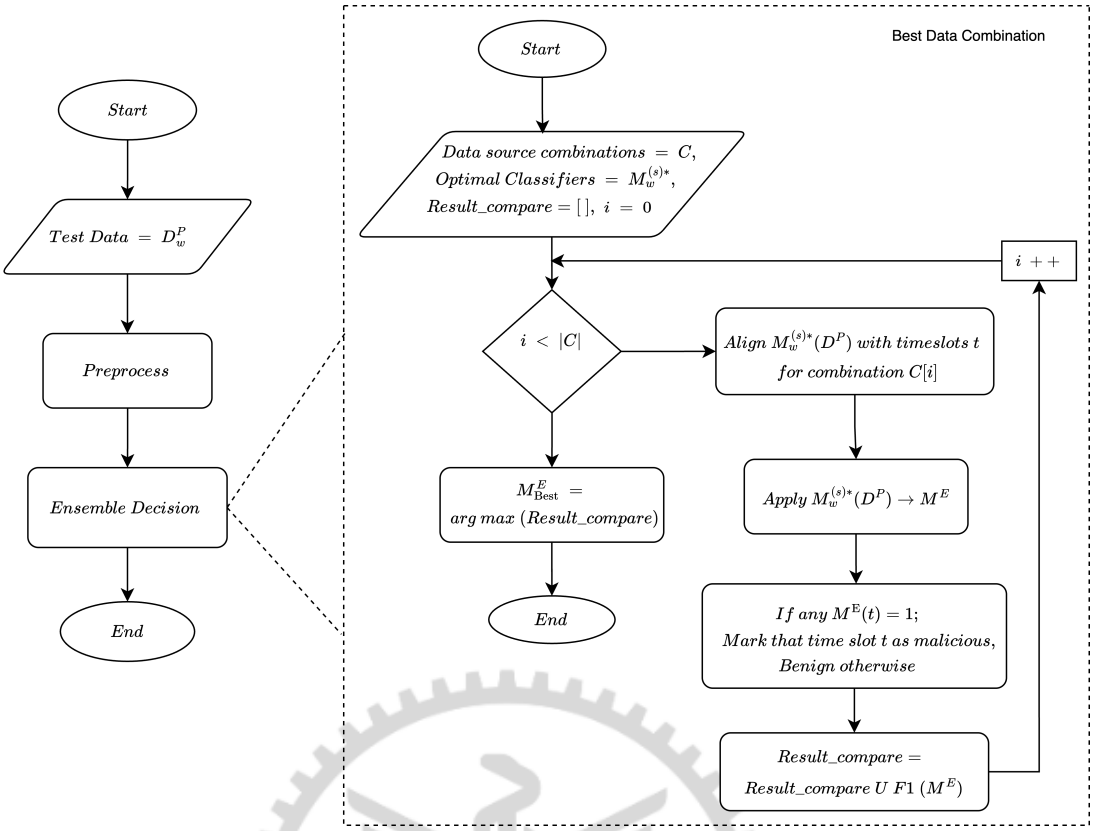


Figure 4.4: Evaluation process for selecting the best-performing data-source combination

4.3.1 Best Data Combination Selection Process

The best-performing combination M^E_{Best} is selected through the following process, shown in Figure 4.4:

1. For each combination C_i, align data across sources by timestamp.
2. Apply optimal classifiers $M_w^{(s)*}$ on the aligned test data.
3. Aggregate predictions: if any source detects an anomaly, label the timestamp as anomaly.
4. Compute the F1-score of each combination.
5. Choose the one with the highest score as M^E_{Best}.

To identify the most effective combination of data sources, we evaluate all valid source combinations. These include:

- Single-source: Sensor, Log, Traffic

- Two-source: Sensor + Log, Sensor + Traffic, Log + Traffic
- Full-source: Sensor + Log + Traffic

4.3.2 Time-Windowed Alignment and Ensemble Decision

In real-world CPS environments, data streams from heterogeneous sources such as sensors, system logs, and network traffic, arrive at different rates and may contain missing or delayed entries. As illustrated in Figure 4.5, each time slot t serves as a common temporal anchor to align entries from different sources. To enable coherent multi-datasource analysis, our framework performs timestamp alignment during preprocessing. Each data stream is aligned independently to a shared set of reference timestamps using a fixed temporal tolerance—chosen to match the shortest reliable reporting interval across sources. This alignment ensures that inputs from different data sources refer to a comparable system state, even though they are processed separately by their respective source-specific models. Missing values are marked and excluded from decision logic, enabling graceful handling of partial observability.

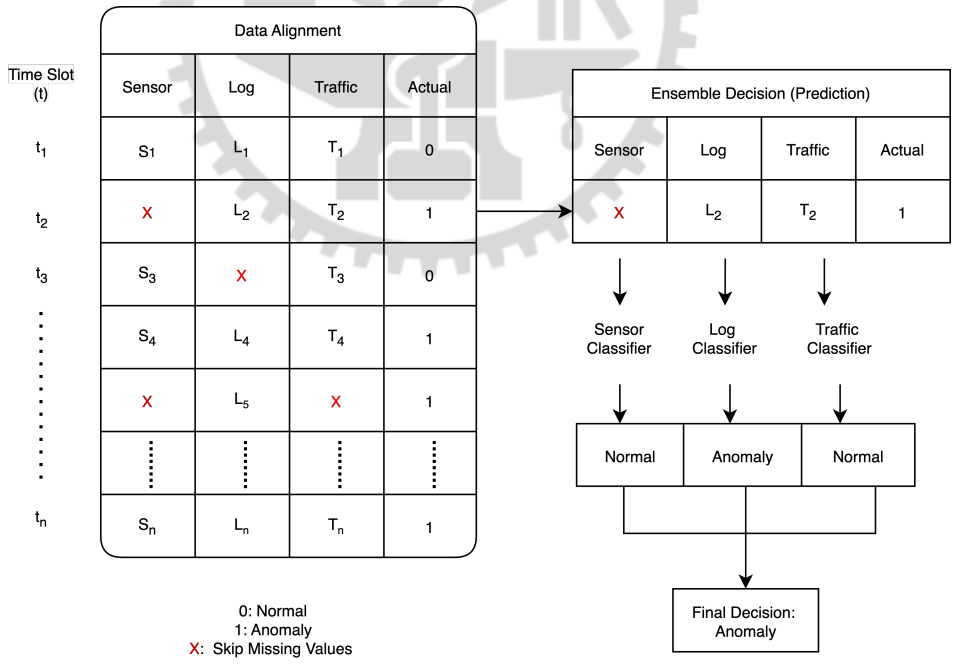


Figure 4.5: Timestamp-aligned data and final ensemble decision logic

The aligned input is processed through a two-stage detection pipeline. In the first stage, source-specific one-class classifiers ($M_w^{(1)}$) detect deviations from normal behavior, having been

trained exclusively on normal samples. If any classifier flags the input as anomalous, the instance proceeds to the second stage, where classifiers ($M_w^{(2)}$), trained on fault-labeled data, categorize the anomaly as either a common system fault or a malicious cyber attack. This modular separation of detection and diagnosis improves both precision and interpretability.

To consolidate predictions across sources, we apply ensemble decision-making. For each aligned timestamp, outputs from sensor, log, and traffic models are combined using an OR-rule strategy. If any source-specific model flags an anomaly (first stage) or detects an event of interest (second stage), that decision is assigned to the timestamp. This conservative approach increases sensitivity to localized or rare anomalies and provides robustness in the presence of missing or noisy data.

For evaluation, we apply a post-processing step using a sliding time window. Predictions are aggregated over fixed-duration intervals (for example, 60 seconds), and if any detected event of interest such as an attack, is present within the window, the entire window is marked accordingly. This window-level labeling reflects operational practices in CPS environments, where even a brief critical event may trigger system-wide alerts or responses. It also improves detection stability by smoothing over transient fluctuations in model outputs.

Overall, the combination of timestamp-aligned preprocessing, two-stage inference, ensemble decision logic, and time window-based evaluation offers a robust and scalable framework for real-time anomaly detection in complex and dynamic CPS deployments.

Chapter 5

Implementation

In this chapter, the implementation of the proposed solution involves the use of open-source tools and libraries, providing a comprehensive experimental setup. This includes the system implementation, the configuration of the testbed, the generation of attack and fault data, and the preparation of data for ML models.

5.1 Open Source Tools and Libraries

The tools and libraries utilized in this thesis play a critical role in various stages of data collection, training, and inference. As detailed in Table 5.1, a range of open-source tools were employed to facilitate different tasks within the system. For data collection, tools like Wireshark [27], Tcpdump [28], and Rsyslog [29] were used to capture and analyze network traffic and system logs. Docker [30] was leveraged for containerization, enabling efficient deployment, isolation, and communication across different components of the system.

Table 5.1: Tools and Libraries

Category	Name	Functionality
Data Collection	Wireshark [27]	Collect and analyze network data.
	Rsyslog [29], Tcpdump [28]	System log collection and network traffic recording.
	Docker [30]	Containerized environment setup, service isolation, and inter-process communication.
Training & Inferencing	Scikit-learn [31]	Classic machine learning algorithms and tools.
	Tensorflow [32]	Deep learning framework for training and inference.
	Matplotlib [33], Pandas [34], Seaborn [35]	Visualization, data manipulation, and statistical graphics.
	NLTK [36]	Natural language processing toolkit for tokenization and text analysis.
	Optuna [37]	Hyperparameter optimization using Bayesian techniques.
	ICSFlow [38]	Industrial control system network traffic feature extraction tool.

For training and inference, ML and deep learning frameworks were used to process the data

and train models. Scikit-learn [31] was employed for general ML tasks, while TensorFlow [32] was used for deep learning models. Data visualization and analysis were carried out using libraries such as Matplotlib [33], Pandas [34], and Seaborn [35]. Text preprocessing tasks for natural language processing were performed with the NLTK library [36], and hyperparameter tuning was handled using Optuna [37]. Network traffic flow extraction was managed with ICS-Flow [38]. These tools collectively form a robust environment for developing and executing the anomaly detection framework in CPS.

5.2 System Implementation

This section describes the implementation of the proposed anomaly detection framework for CPS. The implementation is organized into several key components that address specific aspects of the framework's operation. These components work together to simulate real-world CPS environments and enable the robust evaluation of the detection model's performance.

The first subsection, *Testbed*, details the physical and virtual infrastructure used to emulate a CPS environment. It includes the computing devices and software modules responsible for simulating sensor data, control systems, and attacker behaviors, as well as the use of containerization and preprocessing steps for data collection and model training.

In the *System Fault Generation* subsection, we discuss the methods used to simulate various system faults within the CPS testbed. These faults, which include mechanical failure, hardware degradation, and operational inconsistencies, are essential for testing the framework's ability to distinguish between system faults and cyber attacks.

The *Cyber Attack Generation* subsection outlines the techniques used to simulate cyber attacks within the testbed. This section covers various types of attacks, such as command injections, sensor spoofing, and data manipulation, introduced into the system to evaluate the detection capabilities of the model. By combining fault and attack generation, the system is able to produce a comprehensive dataset for training, testing, and validating the two-stage anomaly detection process.

5.2.1 Testbed

The testbed for our system implementation is structured into several key components, as illustrated in Figure 5.1. The computing device used runs on Ubuntu 20.04, with an Intel Core i7-14700F CPU, an NVIDIA GeForce RTX 4070 GPU, and 32GB of RAM. The testbed is designed to emulate a real-world CPS, featuring several modules that cater to different tasks.

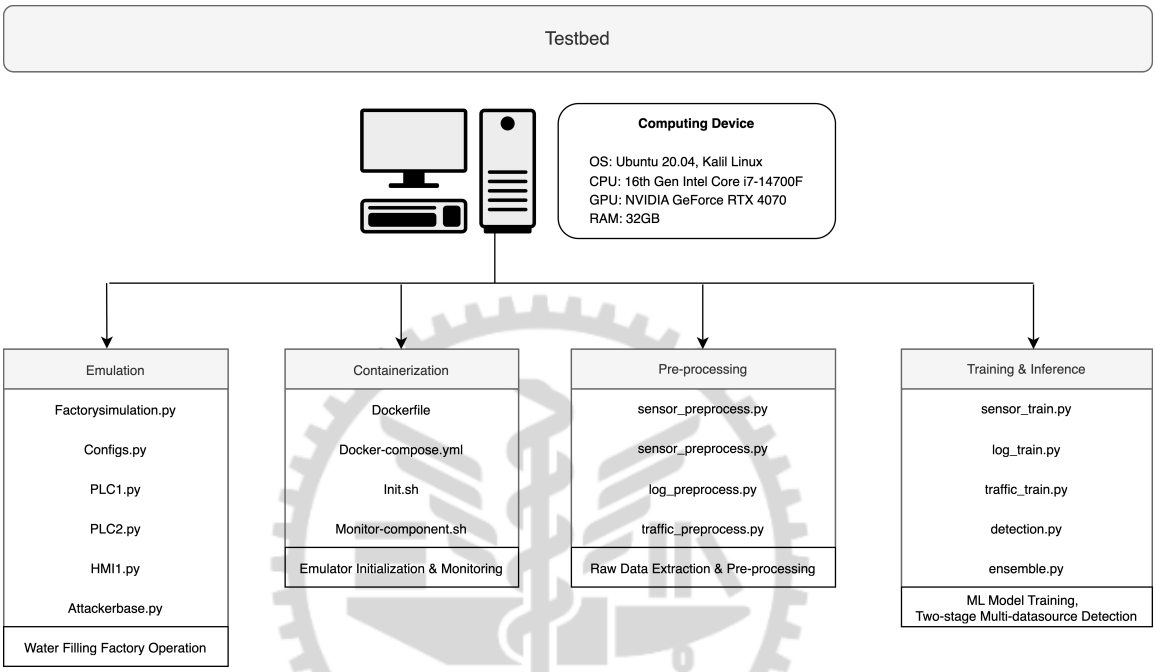


Figure 5.1: Testbed

The Emulation module simulates factory operations through essential Python scripts, such as `Factorysimulation.py`, `Configs.py`, and individual components like `PLC1.py`, `PLC2.py`, `HMI1.py`, and `Attackerbase.py`. These scripts emulate sensor data, control logic, and attacker behaviors, reflecting real-world CPS operations, including anomalies triggered by system faults or cyber attacks.

The Containerization module uses Docker for container management, to ensure consistent packaging and deployment of the system with scripts-`Dockerfile` and `Docker-compose.yml`. Additional scripts, such as `Init.sh` and `Monitor-component.sh`, assist in initializing and monitoring the emulation system, ensuring system stability and scalability.

For the Pre-processing phase, various Python scripts are used to process the sensor data, logs, and traffic data. These steps are critical for normalizing raw data, which is subsequently used

for model training and anomaly detection. The scripts handle the preprocessing of different data types such as `sensor_preprocess.py`, `log_preprocess.py`, and `traffic_preprocess.py` preparing them for training and inference.

The final module, Training and Inference, trains ML models based on the preprocessed data. It includes scripts such as `sensor_train.py` and `log_train.py`, which train individual classifiers for each data source. The `detection.py` script performs multi-datasource detection, while `ensemble.py` aggregates the results for final decision-making. This modular design ensures flexibility, allowing for multi-stage anomaly detection across various data sources.

5.2.2 System Fault Generation

To simulate realistic operational disruptions, the framework incorporates fault injection by modifying the control logic in `PLC1.py` and `PLC2.py`, as illustrated in Figure 5.2. These faults affect sensor inputs, actuator behavior, and internal logic execution, allowing us to evaluate the detection system's ability to distinguish between benign faults and malicious attacks.

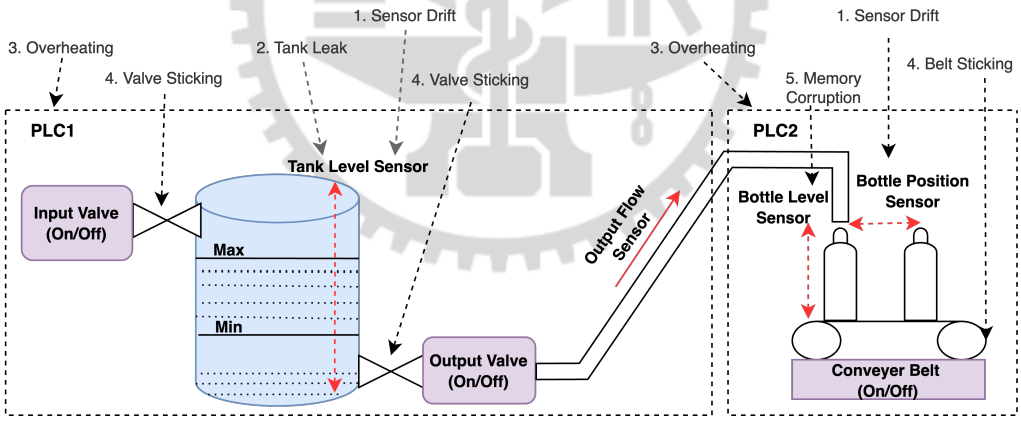


Figure 5.2: System Fault Injection

Five fault types were implemented: (1) Sensor Drift introduces a gradual offset to sensor readings, causing the PLC to misinterpret liquid levels and actuate valves incorrectly. (2) Tank Leak subtracts a small fraction of the tank level (0.01—0.05 per cycle), mimicking an undetected leak that disrupts flow regulation. (3) Overheating adds delays (1—5 seconds) to PLC execution, degrading responsiveness and slowing the conveyor process. (4) Valve Sticking randomly prevents valves from opening or closing, leading to under- or over-filling scenarios. (5) Memory

Corruption injects bit-level errors into control variables, resulting in erratic PLC decisions and actuator misbehavior.

To capture the impact of these faults, sensor readings were logged using Python's logging module, network activity was recorded via Tcpdump, and system logs were collected using Rsyslog. These data sources ensured synchronized observability across physical, network, and system levels. Each fault scenario was activated at runtime through scripted triggers in the PLC configurations, ensuring controlled and repeatable fault conditions within the simulation.

5.2.3 Cyber Attack Generation

To evaluate the resilience of our detection framework against adversarial threats, we simulated a range of cyber attacks targeting key components of the CPS, including PLCs, HMIs, actuators, and network infrastructure. As shown in Figure 5.3, attacks originate from an external actor or a compromised HMI and propagate through the network to disrupt sensor data, actuator behavior, and control commands.

Figure 5.3 illustrates the attack paths, impacted components, and techniques used in our simulation. Red arrows indicate malicious data flows, showing how attacks traverse the network to compromise control logic and physical processes.

Reconnaissance [MITRE T1089]: This phase involves scanning the CPS network to identify active hosts, PLC configurations, and open ports. Tools such as Nmap and Ettercap were used to uncover vulnerabilities and map the network topology, forming the foundation for subsequent attacks.

Denial-of-Service (DDoS) [MITRE T0806]: The attacker floods the network with excessive traffic, particularly targeting PLC communication. Using Ettercap and multi-threaded Nmap, the network is overloaded, preventing real-time command execution and causing delays, dropped packets, and timeouts between HMIs and PLCs.

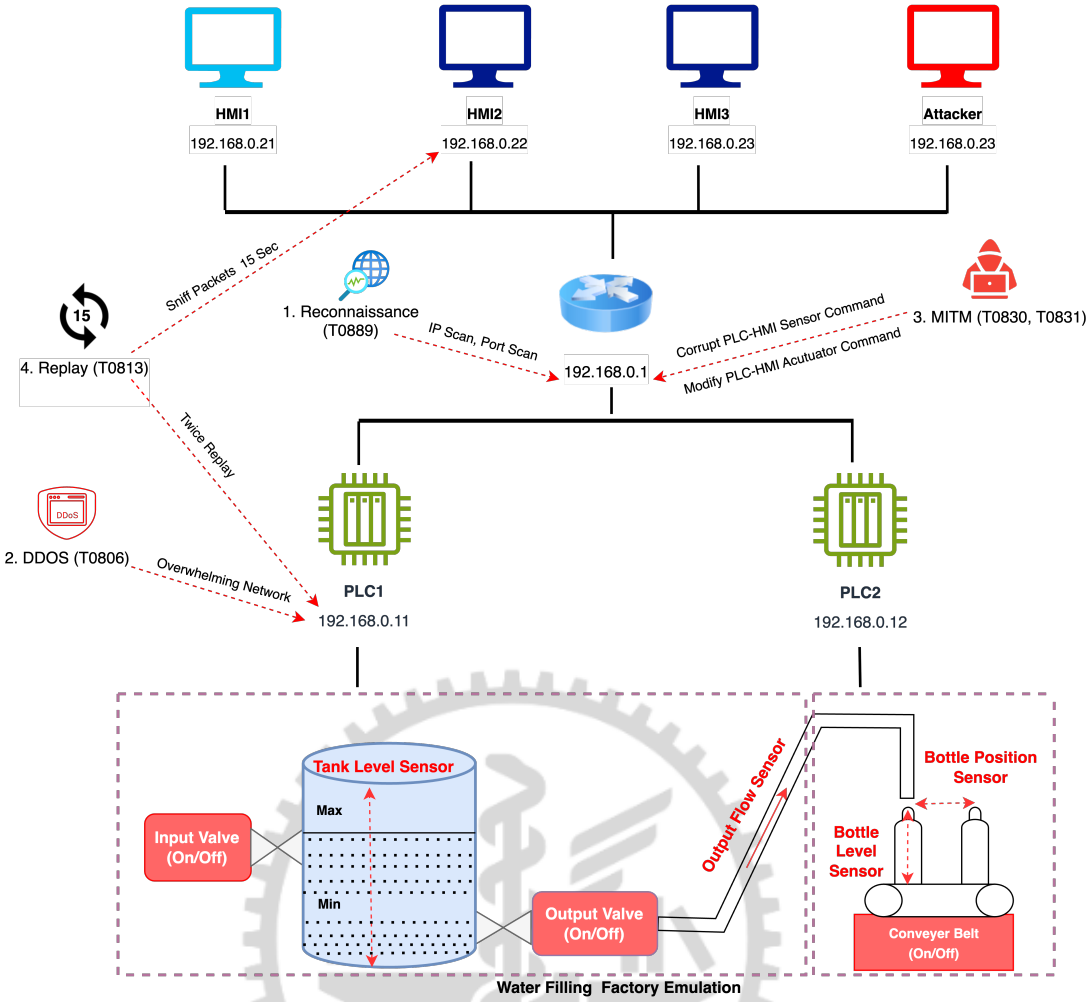


Figure 5.3: Cyber attack injection in the CPS simulation

Man-in-the-Middle (MITM) [MITRE T0830, T0831]: In MITM scenarios, the attacker intercepts communication between PLCs and HMIs. In T0830, sensor data is manipulated mid-transit, while in T0831, forged control messages alter actuator states. These attacks were executed using Scapy and Ettercap, with valid packets first captured using Wireshark and then modified to change sensor values or actuator commands before forwarding.

Replay Attack [MITRE T0813]: The attacker captures valid PLC commands and replays them at inappropriate times to trigger unauthorized actions. Using Wireshark and Scapy, legitimate traffic was recorded and then replayed with altered timing. This led to duplicate actuation or out-of-sequence control execution, such as tank overflow or misaligned conveyor movements.

Overall, the cyber attack scenarios simulate realistic adversarial behaviors and cover multiple tactics from the MITRE ATT&CK framework. The use of tools such as Nmap, Wireshark,

Ettercap, and Scapy enabled the construction, modification, and injection of crafted packets, forming a diverse dataset to evaluate the robustness of our solution.

5.3 Data Preparation

The data preparation process ensures accurate, synchronized collection from multiple sources, forming the foundation for training, validating, and evaluating the anomaly detection models. Raw data is collected from sensor readings, system logs, and network traffic, covering both normal operations and abnormal conditions, including system faults and cyber attacks.

Table 5.2: Sensor Data

PLC	Signals	Type	Range
1	Tank output flow value	Input	Real-time
	Tank level value	Input	Real-time
	Tank level max	Control	On/Off/Auto
	Tank level min	Control	On/Off/Auto
	Tank input valve mode	Output	On/Off
	Tank output valve mode	Output	On/Off
	Tank input valve state	Output	On/Off
	Tank output valve state	Output	On/Off
2	Bottle level value	Input	Real-time
	Bottle distance to filler value	Input	Real-time
	Bottle level max	Control	On/Off/Auto
	Conveyor belt engine mode	Control	On/Off/Auto
	Conveyor belt engine state	Output	On/Off

Sensor data, as described in Table 5.2, includes key signals such as tank levels, valve states, and conveyor belt statuses. The raw data is first cleaned and normalized to eliminate inconsistencies. Missing values are imputed using statistical methods such as mean or median replacement to ensure continuity. Feature extraction is then applied to compute derived attributes like rate of change or deviation from baseline, which enhance anomaly detection. These features are subsequently scaled using Min-Max normalization to maintain uniform input ranges across the dataset.

Table 5.3: System Log Template

Log Template	Description
[PLC1] CRITICAL Processing delayed by <*> seconds	Triggered when processing is delayed
[PLC1] WARNING Tank level reduced to <*>	Issued when tank level is too low
[PLC1] CRITICAL Tank output flow set to <*>	Logged when output flow is forcibly changed

Table 5.4: Network Traffic Flow Features

Category	Name	Functionality
Data Volume	SentBytesAverage	Average bytes sent per flow
	ReceivedBytesAverage	Average bytes received per flow
	SentPayloadSizeAverage	Average payload size in sent packets
	ReceivedPayloadSizeAverage	Average payload size in received packets
TCP Packet Rates	SentFINPacketRate	Rate of FIN packets sent
	ReceivedFINPacketRate	Rate of FIN packets received
	SentSYNPacketRate	Rate of SYN packets sent
	ReceivedSYNPacketRate	Rate of SYN packets received
	SentRSTPacketRate	Rate of RST packets sent
	ReceivedRSTPacketRate	Rate of RST packets received
Acknowledgment	SentACKPacketRate	ACK packets sent rate
	ReceivedACKPacketRate	ACK packets received rate
	SentACKDelayMaximum	Maximum delay in ACK from sender
	ReceivedACKDelayMaximum	Maximum delay in ACK from receiver
Time & Interval	SentInterPacketIntervalAverage	Avg. inter-packet time for sent packets
	ReceivedInterPacketIntervalAverage	Avg. inter-packet time for received packets
	FlowStartTimestamp	Timestamp when flow started
Traffic Summary	SentPacketCount	Number of packets sent
	ReceivedPacketCount	Number of packets received
	TransportProtocol	Network protocol used (e.g., TCP)
TTL	SentTTL	Average TTL of outgoing packets
	ReceivedTTL	Average TTL of incoming packets
TCP Window	SentTCPWindowSize	TCP window size from sender
	ReceivedTCPWindowSize	TCP window size from receiver

System logs are collected in template-based formats, as shown in Table 5.3. These logs are cleaned to remove irrelevant characters and structured into consistent patterns. The cleaned text is vectorized using Term Frequency-Inverse Document Frequency (TF-IDF), converting each log entry into a numerical vector that captures the significance of log terms. This representation enables the model to interpret log content quantitatively for anomaly detection.

For network traffic, communications between HMIs, PLCs, log servers, and external endpoints are recorded. Raw packets are captured using Wireshark and Tcpdump, then processed with ICSFlow to generate flow-level representations. As detailed in Table 5.4, features such as packet count, byte count, flow duration, and frequency are extracted. Noise such as ARP, broadcast, and DNS traffic is filtered out, and timestamps are normalized to enable cross-source temporal alignment.

Once preprocessing and feature scaling were completed across all data sources, the dataset was partitioned into three subsets: 70% for training unsupervised models, 10% for validation and hyperparameter tuning, and 20% for testing. This ensured a consistent and reproducible pipeline for anomaly detection in CPS.

After preprocessing the sensor, log, and traffic datasets, we visualized their structure using 2D t-SNE plots to analyze the distribution and overlap among normal, fault, and attack classes. As shown in Figure 5.4, each subplot corresponds to one data source. In the sensor data (Figure 5.4a), the three classes formed compact clusters, but there was notable overlap: attack samples partially intersected with both normal and fault regions. This made classification difficult using sensor data alone, particularly in single-stage settings.

(a) Sensor Data (b) Log Data (c) Traffic Data

Figure 5.4: 2D t-SNE visualization showing class separation across data sources

In contrast, the log (Figure 5.4b) and traffic (Figure 5.4c) data reveal more separable attack patterns, though fault and normal samples often intermingle. This suggests that logs and traffic can better isolate attack behavior, while sensor data captures physical deviations more precisely but may confuse faults and attacks due to behavioral mimicry.

These observations reinforce two key insights. First, single-source models suffer from limited perspective: one modality alone cannot consistently distinguish all three classes due to overlaps in feature space. Second, single-stage models trained to directly classify all categories are prone to confusion, particularly where fault and attack symptoms intersect.

Our proposed two-stage, multi-datasource framework directly addresses these issues. first stage uses one-class learning to model only the normal class well-defined and abundant—making it highly sensitive to deviations regardless of their type. The second stage then classifies the detected anomalies into fault or attack using only fault-labeled data, thus removing dependency on volatile and incomplete attack labels.

Moreover, by aligning and aggregating outputs from all three data sources, the framework compensates for ambiguity in any single source. For instance, when sensor data fails to distinguish attacks clearly, distinctive patterns from logs or traffic can still guide the final decision. Even though sensor-based attacks may appear more separated from faults in this dataset, such separation is often artificial in real-time CPS where adversaries mimic faults to avoid detection. This further justifies the need to explicitly differentiate faults from attacks, as our architecture is designed to do.

Chapter 6

Evaluation and Results

This chapter details the ML models specifications and settings required to construct the experiment. Lastly, the experiment's results and analysis are presented.

6.1 Experimental Setup and Parameters

To evaluate the proposed two-stage anomaly detection framework, we implemented and tuned four types of unsupervised ML models: OCSVM, Isolation Forest, Local Outlier Factor (LOF), and Autoencoder. Each model was trained and evaluated independently on three data sources sensor signals, system logs, and network traffic across two detection stages. These stages correspond to anomaly detection (first stage) and fault/attack detection (second stage). All configurations were optimized specifically for each model and data type to ensure reliable performance across heterogeneous sources.

Hyperparameter tuning was performed using Optuna [37], an automated optimization framework that applies Bayesian optimization with pruning strategies to explore the hyperparameter space efficiently. For each model, the search aimed to maximize the F1-score on the validation set. The tuning parameters included: kernel type, ν, and γ for OCSVM; number of estimators and contamination rate for Isolation Forest; number of neighbors and contamination rate for LOF; and encoder-decoder architecture (layer width) with optimizer: adam(lr=0.001)and loss: mse for Autoencoders. Trials were pruned early when validation performance failed to improve, reducing overfitting and computation time.

The final selected hyperparameters for each model and modality, derived from Optuna's optimization process, are detailed in Table 6.1. These configurations served as the basis for model training in both the first and second stages of the detection pipeline.

Table 6.1: Hyperparameter Settings of ML Classifiers

Data	Stage	OCSVM	Isolation Forest	LOF	Autoencoder
Sensor	1	Sigmoid, Nu=0.32, Gamma=Scale	Estimators=143, Cont=0.13	Neighbors=10, Cont=0.03	14 → 59→ 32 → 59 → 14, Sigmoid | Total Params: 5,592
	2	Sigmoid, Nu=0.44, Gamma=Scale	Estimators=66, Cont=0.18	Neighbors=19, Cont=0.03	14 → 64 → 32 → 64 → 14, Sigmoid | Total Params: 6,062
Log	1	Sigmoid, Nu=0.20, Gamma=Auto	Estimators=109, Cont=0.23	Neighbors=38, Cont=0.06	100 → 64 → 48 → 64 → 100, Sigmoid | Total Params:19,220
	2	Sigmoid, Nu=0.08, Gamma=Scale	Estimators=171, Cont=0.34	Neighbors=37, Cont=0.12	100 → 72 → 36 → 72 → 100, Sigmoid | Total Params: 19,864
Traffic	1	Sigmoid, Nu=0.16, Gamma=Auto	Estimators=199, Cont=0.35	Neighbors=50, Cont=0.10	23 → 90 → 45 → 90 → 23, Sigmoid | Total Params: 12,488
	2	Sigmoid, Nu=0.18, Gamma=Scale	Estimators=112, Cont=0.37	Neighbors=22, Cont=0.06	23 → 44 → 22 → 44 → 23, Sigmoid | Total Params: 4,093

6.2 Datasource vs. Detection Stage

This section analyzes how detection performance varies across combinations of data sources and detection strategies. Figure 6.1 shows F1-scores for seven configurations, including both single- and multi-source inputs under single-stage and two-stage architectures. The X-axis lists each configuration (S: Sensor, L: Log, T: Traffic, and their combinations), and the Y-axis shows the corresponding F1-score. Light green bars represent single-stage results, while dark green bars indicate two-stage performance.

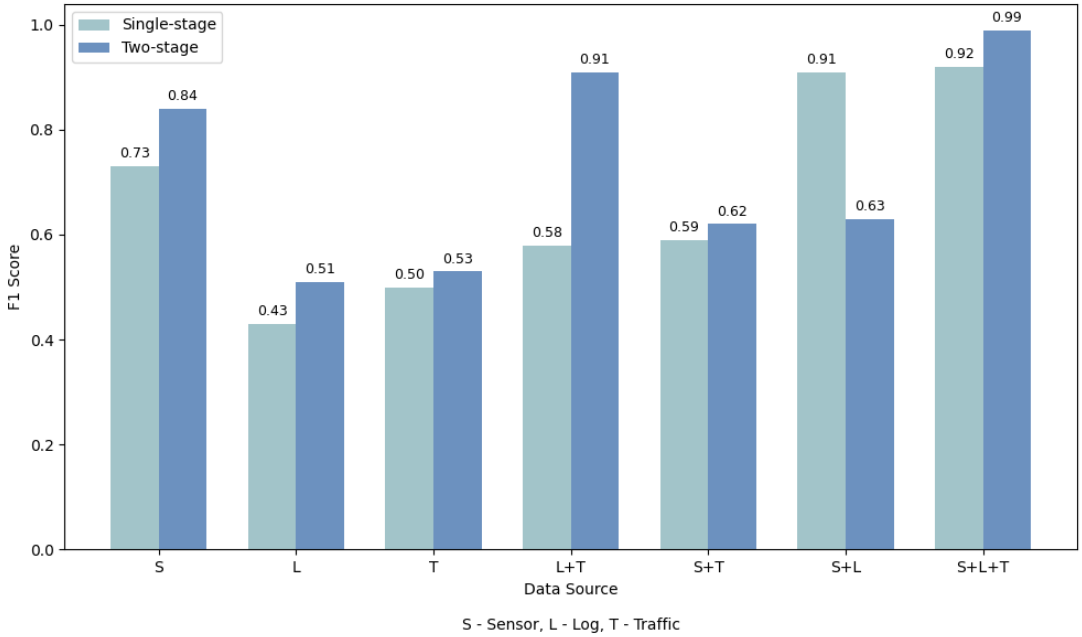


Figure 6.1: F1-score comparison for data sources and detection strategies

We begin with our proposed method two-stage detection with full multi-datasource input (S+L+T) which achieves the highest F1-score of 0.99. This improvement results from two complementary mechanisms. First, the alignment of heterogeneous data captures a more complete

picture of CPS behavior. Second, the two-stage architecture modularizes the task: first stage isolates deviations from normal patterns using only normal data, while second stage classifies these deviations into faults or attacks using only fault-labeled data. This staged reasoning reduces confusion between overlapping behaviors and allows robustness to unseen attacks.

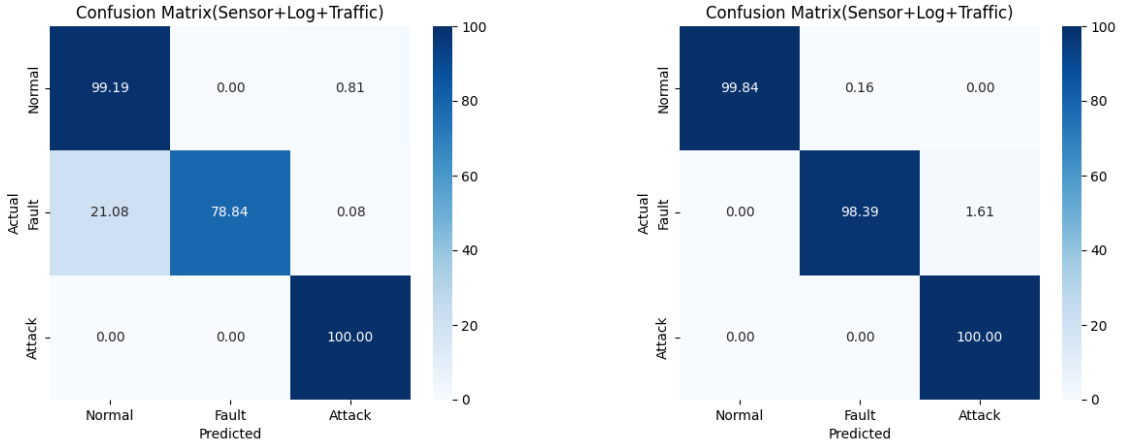
Interestingly, for the Sensor+Log (S+L) combination, the single-stage method slightly outperforms the two-stage variant (0.91 vs. 0.63 F1 score). This reversal likely stems from the information redundancy between sensor and log data, which the single-stage method can exploit more directly. In contrast, the two-stage architecture may struggle when log data introduces noise or inconsistent temporal cues that interfere with clean anomaly separation in first stage. Without the stabilizing context of traffic data, the second stage receives less reliable anomaly inputs, leading to reduced final classification accuracy. This highlights that two-stage detection is more sensitive to the quality and complementarity of input sources, and benefits most from full multi-source input.

In contrast, the single-stage multi-datasource baseline achieves an F1 score of 0.92. While it benefits from source diversity, it struggles to separate faults from normal data due to overlapping symptoms and the lack of intermediate anomaly filtering. It is also more reliant on complete and balanced attack labeling, which limits its robustness in dynamic environments.

Sensor-only detection using the two-stage method yields an F1-score of 0.84, which outperforms any single-stage variant. This demonstrates the architectural advantage of decomposing the detection process, even without log or traffic support. However, without those additional sources, fault/attack disambiguation remains weaker. Two-stage method using only log or traffic data yield F1-scores of 0.51 and 0.53. These sources alone are sparse and noisy, and their temporal inconsistencies limit their detection reliability especially without the physical grounding provided by sensor input.

The weakest performance comes from one-stage detection with single-source inputs. F1-scores drop to 0.43 for logs and 0.50 for traffic. These methods fail to separate faults from attacks

due to similar behavioral traces. Even sensor-only single-stage detection (0.73) underperforms its two-stage version by 11%, confirming the benefit of stage separation.



Single-stage Two-stage

Figure 6.2: Confusion matrices of best combination (Sensor + Log + Traffic)

To better understand the anomaly detection behavior of the best-performing single-stage and two-stage multi-datasource approaches, we examine their confusion matrices in Figure 6.2. The figure illustrates how each approach handles the distinction between normal, fault, and attack instances. The two-stage approach (right) is better aligned with practical CPS scenarios, where attack patterns are constantly evolving and labeled attack examples may not be available. This approach is trained only with normal data in the first stage (for one-class anomaly detection) and with fault-labeled data in the second stage. Attack data is never seen during training and is used solely during evaluation. Despite this, the two-stage model successfully detects all attacks and faults without misclassification. The first stage flags deviations from normal behavior, and the second stage further distinguishes between known faults and unknown anomalies effectively identifying attacks. This separation of concerns simplifies the decision boundaries, reduces fault-normal confusion, and avoids overfitting to sparse or static attack signatures. As a result, the two-stage approach achieves a 7% improvement in F1 score and demonstrates greater robustness to unseen and evolving attacks for real-world CPS.

In contrast, the single-stage approach (left) achieves perfect attack detection but exhibits significant confusion between fault and normal data 21.08% of fault instances are misclassified

as normal. This misclassification stems from overlapping patterns in log and traffic data, as visualized in Figures 5.4b and 5.4c, which obscure the boundary between fault and normal behavior. Although the single-stage multi-datasource approach detects all attacks, it was trained using attack-labeled data, and both training and test data were drawn from the same distribution an assumption that often breaks down in real-world CPS settings.

Table 6.2: Testing Time Comparison between Single-stage and Two-stage Methods

Method	**Stage**	**Sensor (S)**	**Log (L)**	**Traffic (T)**	**S + L + T**
Single-stage (LightGBM)	Total Only	0.184 ± 0.087	0.164 ± 0.071	0.148 ± 0.064	0.170 ± 0.086
Two-stage (OCSVM)	First Stage	3.710 ± 0.002	5.451 ± 0.005	2.079 ± 0.003	5.657 ± 0.023
	Second Stage	2.578 ± 0.008	3.959 ± 0.004	3.803 ± 0.001	4.709 ± 0.012

To evaluate the practical viability of the proposed detection strategies in real-time CPS settings, we compare their inference latency across different data sources, as shown in Table 6.2. The two-stage method incurs higher latency, with inference times ranging from 2.079 to 5.451 seconds depending on the datasource and processing stage. This overhead comes from the use of OCSVMs in both stages: first to detect anomalies and then to classify them as faults or attacks. OCSVMs rely on computationally intensive kernel operations involving high-dimensional support vectors, making them slower at inference. Since both stages operate sequentially and independently, the total latency reflects the additive cost of these two passes.

In contrast, the single-stage approach, implemented using a gradient-boosted decision tree ensemble (LightGBM), demonstrates consistently low latency—under 0.2 seconds per input across all data sources. This efficiency is due to its tree-based architecture, fixed-depth traversal, and high degree of parallelism. While the two-stage method introduces greater computational demand, the trade-off is justified by its enhanced detection robustness. Unlike the single-stage method, which requires attack-labeled training data and assumes stable data distributions, the two-stage approach operates effectively without ever encountering attack data during training. This allows it to detect previously unseen or evolving attacks—an essential capability in dynamic and unpredictable CPS environments. Crucially, the overall latency remains within practical limits (≤10 seconds), making the approach suitable for real-time applications where rapid but not necessarily instantaneous responses are acceptable.

Specifically, in the full multi-datasource (S+L+T) configuration, the two-stage method reaches a combined inference time of 4.709 seconds, substantially higher than the 0.170 seconds recorded by the single-stage baseline. However, this added latency is offset by a marked improvement in detection performance (F1-score: 0.99 vs. 0.92), underscoring the benefit of incorporating richer input and staged reasoning. Additionally, the lower latency observed in the second stage suggests that once anomalies are filtered, final classification becomes computationally more efficient. These findings highlight that the two-stage architecture, though more resource-intensive, remains both scalable and deployable in time-sensitive industrial CPS environments demanding robust detection.

In summary, the two-stage multi-datasource approach delivers robust and deployment-ready anomaly detection for CPS, achieving an F1-score of 0.99 with sensor-log-traffic input. By aligning heterogeneous data and separating detection into two stages, it effectively handles overlapping behaviors without relying on attack labels. Unlike the single-stage approach which depends on labeled attacks and misclassifies 21.08% of faults as normal the two-stage framework achieves perfect separation, highlighting its robustness against unknown attacks. Though the use of kernel-based OCSVMs in both stages introduces moderate computational overhead (2—5 seconds per input), this cost is outweighed by the approach's resilience to unseen and evolving attack patterns. Unlike the single-stage baseline, which benefits from training-test distribution alignment, the two-stage method requires no attack labels and still detects all attack samples demonstrating robustness under data uncertainty, a critical requirement in real-world CPS.

6.3 Effect of Time Windows

In multi-datasource anomaly detection for CPS, selecting an appropriate time window size is critical for achieving reliable detection performance. Sensor, log, and traffic data streams operate at different temporal granularities Sensor and traffic data streams operate at high temporal resolution, often generating events at millisecond to second intervals, while log data remains sparse and irregular. This disparity in frequency makes time window selection a critical factor for aligning heterogeneous signals into a coherent temporal snapshot of system behavior. As

such, the window size determines how well data from different sources align temporally to form a coherent snapshot of system behavior. Investigating this effect helps reveal how different detection approaches exploit or depend on temporal context, especially under conditions of data sparsity or event delay common in real-world CPS environments.

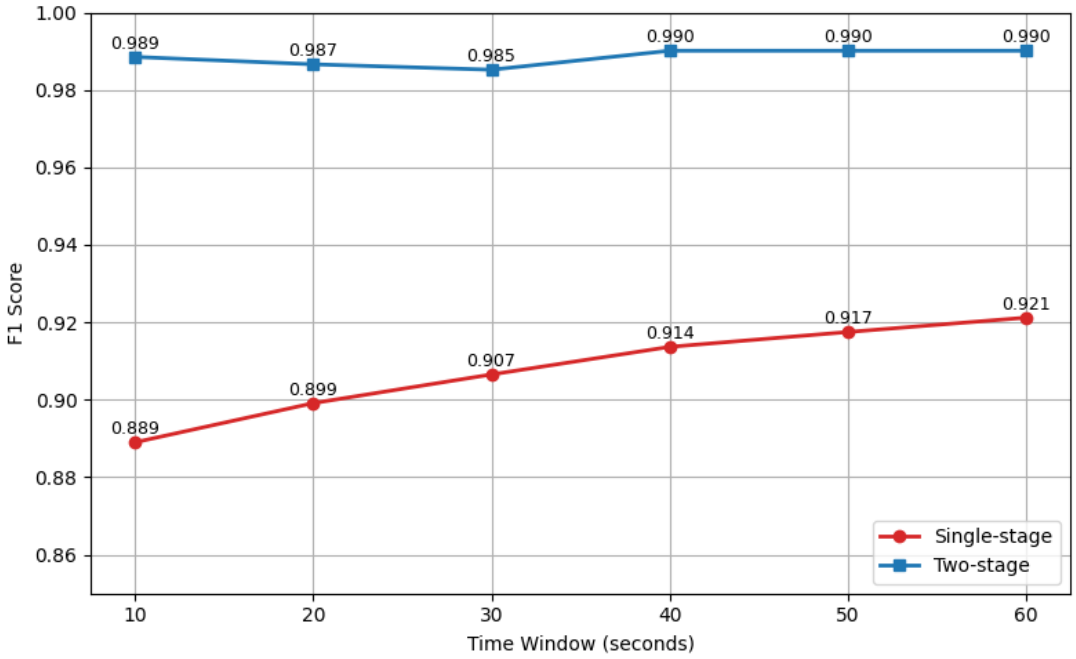


Figure 6.3: Effect of Time Window Size on F1 Score

To quantify the impact of time window size on detection performance, we evaluate how F1 scores change across increasing window lengths for both single-stage and two-stage approaches, as shown in Figure 6.3. The two-stage multi-datasource approach reaches an F1 score of 0.99 by the 40-second time window and maintains this performance with larger windows. This early convergence reflects the architectural strengths of the approach. The first stage, trained solely on normal data, robustly flags anomalous deviations, while the second stage distinguishes between faults and attacks using only fault-labeled samples. Because the anomaly detection stage operates independently of fault/attack class boundaries, it requires less contextual buildup to trigger a meaningful response. Larger windows mainly reinforce redundancy rather than improve detection performance, enabling the approach to remain both effective and timely.

In contrast, the single-stage multi-datasource approach improves more gradually from an F1 score of 0.889 at 10-second windows to 0.921 at 60 seconds. Without an intermediate anomaly

detection layer, this approach must simultaneously learn to detect deviations and classify them into normal, fault, or attack categories within a single model structure. Consequently, it depends more heavily on longer time windows to gather enough information about how the system's behavior evolves over time especially from sparse sources like logs to resolve ambiguities caused by overlapping behavioral patterns. This need to observe longer periods of system activity in order to detect anomalies accurately makes the single-stage approach less efficient in environments where timing is critical or data from different sources arrives inconsistently.

Although some may suggest further increasing the window size for the single-stage approach to boost its performance, this comes with diminishing returns and potential drawbacks. Larger windows demand greater memory and computational resources and can introduce latency, which is undesirable in real-time or resource-constrained environments. Moreover, extending the window beyond 60 seconds may amplify redundancy without significantly improving detection accuracy. To balance performance, responsiveness, and system overhead, we restrict the evaluation range from 0 to 60 seconds in 10-second increments capturing the essential performance trends while remaining practical for deployment in CPS.

In summary, while longer windows benefit both approaches by enhancing temporal alignment and feature completeness, the two-stage approach achieves high robustness with shorter observation periods. This reflects its architectural efficiency and its ability to make reliable decisions with limited but focused input. Such characteristics are particularly valuable in real-time CPS settings, where fast and reliable anomaly detection under partial observability is a core requirement.

6.4 Learning Models for Two-stage & Single-stage Detection

This section compares the performance and training cost of models used in both two-stage and single-stage detection pipelines. In the two-stage setup, four unsupervised models—OCSVM, Isolation Forest, LOF, and Autoencoder, are evaluated across two phases: anomaly detection and fault/attack separation, as shown in Figures 6.4 and 6.5. In the single-stage setting, four

supervised models—LightGBM, Decision Tree, Random Forest, and XGBoost, are trained to directly classify each instance as normal, fault, or attack (Figure 6.6). Performance is assessed using F1-score across three data sources: sensor, log, and traffic. The results are interpreted based on the learning characteristics of each model and the underlying class separability, as illustrated in the t-SNE plots and class distribution maps (Figure 5.4). In addition to detection accuracy, we compare the computational efficiency of models through average training time, summarized in Table 6.3. This joint analysis informs the selection of models that balance accuracy, robustness, and efficiency for real-time CPS deployment.

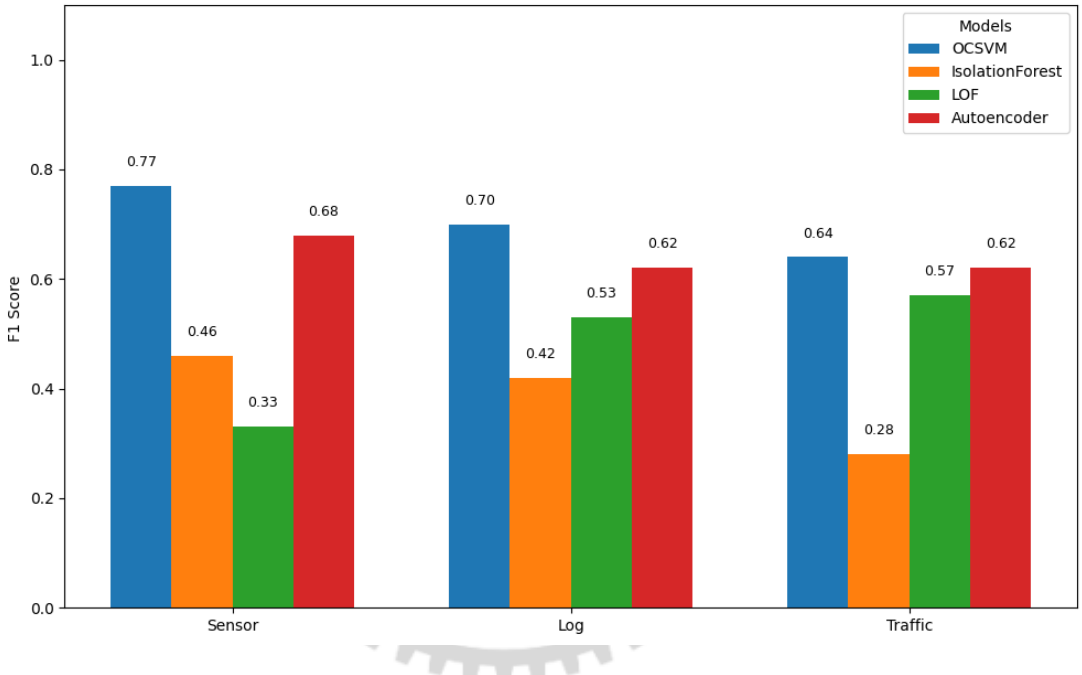


Figure 6.4: Two-stage Detection Performance Comparison Across Models - First Stage

In the first stage of the two-stage detection pipeline, models are trained solely on normal data and evaluated on a binary classification task: distinguishing normal from anomalous behavior. As shown in Figure 6.4, OCSVM consistently achieves the highest F1-scores—0.77 for sensor, 0.70 for log, and 0.64 for traffic due to its kernel-based formulation, which effectively captures tight decision boundaries around normal samples. Autoencoders also perform well on sensor data (0.68), where reconstruction of structured input is reliable, but their accuracy drops on traffic and log data due to input sparsity and irregularity. LOF underperforms on sensor data (0.33), where local density differences are minimal, though it performs moderately on traffic (0.57), which exhibits more variation. Isolation Forest shows the weakest performance overall

(e.g., 0.28 on traffic), as its random partitioning strategy struggles with sparse, high-dimensional, and imbalanced data distribution.

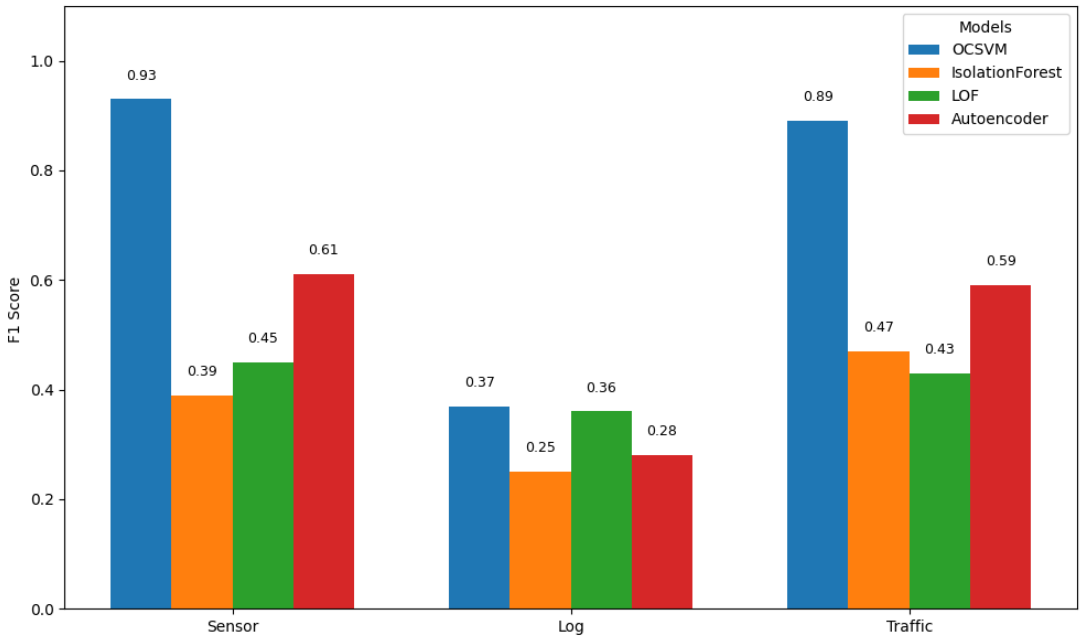


Figure 6.5: Two-stage Detection Performance Comparison Across Models - Second Stage

In second stage, the models are trained only on fault-labeled data and evaluated on the task of classifying anomalies as either faults or attacks. The class imbalance is high here faults dominate while attacks are rare and variable. Figure 6.5 shows the F1-score comparison, with the X-axis representing data sources and the Y-axis showing detection F1-scores. OCSVM again performs best, achieving 0.93 on sensor and 0.89 on traffic, thanks to its robustness in learning from stable fault patterns. Its performance on logs drops to 0.37 due to noisy semantics and fewer well-aligned fault logs. Autoencoders provide moderate performance (for instance, 0.61 on sensor), but underperform on logs (0.28) where reconstruction fails to discriminate between fault and attack. LOF and Isolation Forest show the lowest F1-scores across all sources, especially on traffic and logs, due to poor boundary learning on limited, imbalanced fault data.

As shown in Figure 6.6, the single-stage detection approach uses supervised learning models to directly classify input instances into normal, fault, or attack classes. This unified pipeline eliminates the need for separate anomaly detection and fault/attack discrimination, simplifying deployment and reducing training overhead. However, its effectiveness depends heavily

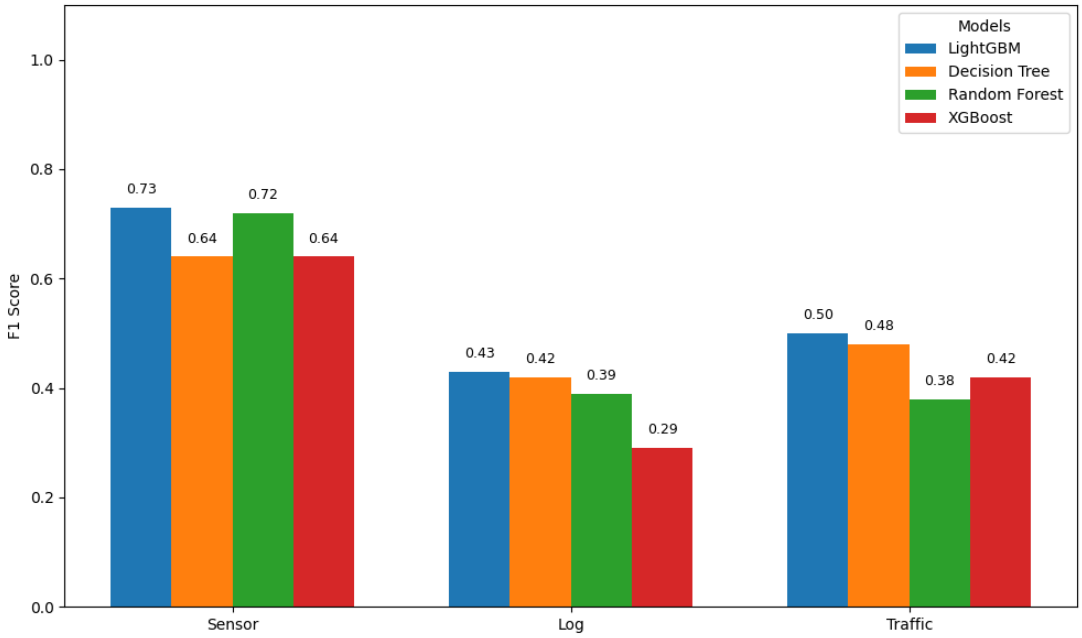


Figure 6.6: Single-stage Detection Performance Across Models

on the separability of class distributions. As seen in Figure 5.4, sensor data forms compact, well-separated clusters, enabling high F1-scores across models (e.g., 0.73 for LightGBM and 0.72 for Random Forest). Log data, by contrast, is sparse and interleaved, leading to significantly lower performance (e.g., 0.29 for XGBoost) due to its high class overlap and limited structure. Traffic data lies between the two extremes, yielding moderate scores (up to 0.50 with LightGBM). LightGBM consistently outperforms others due to its gradient-based optimization, class-weighting, and efficient handling of feature sparsity. Decision Trees remain competitive, especially on traffic data (0.48), by prioritizing strong splits even in noisy inputs. Random Forest performs well only on structured sensor data, where its ensemble averaging supports stable generalization. XGBoost, though powerful on clean numerical data, underfits on log data due to unreliable second-order gradient estimates on sparse inputs. These results show that while single-stage models are computationally efficient and practical for real-time CPS deployment, their accuracy deteriorates in scenarios with overlapping or poorly separated class distributions, where two-stage detection methods offer stronger resilience.

To evaluate the trade-off between detection performance and computational cost, we compare the training times of various models across different stages and data sources. Table 6.3 presents the average training time (in minutes) for each model in both two-stage and single-stage de-

tection settings. Among the two-stage method, Isolation Forest and LOF are the fastest (under one minute), but their limited detection accuracy renders them unsuitable for critical CPS applications. OCSVM requires longer training, particularly on log data (18.63 minutes), due to the computational overhead of kernel-based learning on high-dimensional inputs. However, its second-stage training, performed only on labeled fault data—is more efficient. Autoencoders exhibit the longest training times (e.g., 11.5 minutes on logs), primarily due to the complexity of learning deep representations. Despite their modeling capacity, their moderate accuracy and high training cost limit their applicability in time-sensitive environments.

Table 6.3: Training Time of ML Models (in minutes)

Method	Models	Stage	Sensor	Log	Traffic
Two-stage	OCSVM	1	0.597	18.628	5.133
		2	0.013	7.630	1.087
	Isolation Forest	1	0.009	0.026	0.018
		2	0.005	0.029	0.010
	LOF	1	0.035	1.611	0.082
		2	0.011	0.230	0.035
	Autoencoder	1	1.890	11.497	2.784
		2	0.845	3.040	2.562
Single-stage	LightGBM	Total Only	0.100	5.561	0.029
	Decision Tree		0.001	0.051	0.005
	Random Forest		0.054	12.136	0.048
	XGBoost		0.263	0.492	0.305

For single-stage detection models, training times vary with model complexity and data characteristics. Decision Trees are the most efficient (e.g., 0.001 minutes on sensor data), owing to their simple, non-iterative structure and minimal parameter tuning. LightGBM offers better efficiency than most ensemble methods by leveraging histogram-based feature binning and leaf-wise tree growth. In contrast, Random Forest incurs higher training time (e.g., 12.14 minutes on log data), as it constructs numerous trees across bootstrapped samples. XGBoost, while generally faster than Random Forest, remains slower than LightGBM due to its use of second-order gradients and regularization techniques. Overall, single-stage models present lower training overhead and simpler deployment pipelines, making them attractive for time-constrained CPS environments. However, they may offer reduced robustness against novel attacks compared to two-stage approaches, which benefit from more granular separation of detection tasks.

In summary, OCSVM demonstrates the strongest performance in the two-stage pipeline by learning stable, global decision boundaries that generalize well to unseen attack behaviors—especially on structured sensor data. While Autoencoders show moderate success on consistent inputs, they struggle with sparsity and noise. LOF and Isolation Forest offer fast training but lack robustness in complex CPS scenarios. In the single-stage setting, LightGBM consistently achieves the highest F1-scores across data sources due to its ability to handle class imbalance and sparse features efficiently. Overall, model effectiveness is closely tied to data characteristics: OCSVM is best suited for high-assurance two-stage detection, while LightGBM offers a strong balance of accuracy and efficiency for single-stage, real-time CPS deployment.

Chapter 7

Conclusion and Future Work

This final chapter summarizes the findings of this thesis, highlighting the contributions and evaluation outcomes of the proposed anomaly detection framework. It also outlines potential directions for future improvements and research extensions.

7.1 Conclusion

This thesis proposed a two-stage, multi-datasource anomaly detection framework for CPS, specifically designed to distinguish between cyber attacks and system faults. By aligning and processing sensor, log, and traffic data independently and synchronously, the framework captures a more holistic view of system behavior. Experimental results demonstrate that the proposed framework achieves a substantial F1-score improvement over conventional configurations: 15% higher than the best single-source two-stage method, 27% higher than the single-source single-stage baseline, and 7% higher than the multi-datasource single-stage approach, culminating in an overall F1-score of 0.99. These gains are attributed to contextual diversity across data sources and the decoupling of detection and classification into two dedicated stages. first stage uses one-class training on normal data to detect deviations, while second stage leverages only fault-labeled samples to distinguish attacks from faults removing dependence on volatile attack labels.

Evaluation of different time window sizes further confirmed the robustness and efficiency of the two-stage approach. It achieved its highest F1-score at the 40-second window and remained stable beyond that, demonstrating its ability to deliver reliable performance even with shorter input durations. These results reinforce the suitability of the two-stage approach for real-time CPS applications, where timely and accurate detection with limited input is essential.

Among the learning models evaluated, OCSVM consistently performed best across both stages and all data sources. Its ability to model stable normal behavior enabled it to detect unseen attacks effectively, while maintaining a practical inference latency of under 6 seconds per input. In contrast, faster alternatives such as Isolation Forest and LOF struggled with the sparsity and high dimensionality of CPS data, underscoring the trade-off between computational speed and detection robustness. Overall, the proposed solution achieves high detection accuracy, strong resilience to unseen threats, and practical feasibility for real-world CPS deployment.

7.2 Future Work

Future work will focus on extending the proposed two-stage multi-datasource approach toward real-time, adaptive anomaly detection in CPS by integrating reinforcement learning (RL) for online model adaptation and preparing the system for deployment on edge devices. An RL-based controller can learn to adjust model parameters such as detection thresholds or update strategies based on feedback from detection outcomes, allowing the system to adapt over time without requiring full retraining. To ensure safe and reliable behavior, these adaptive mechanisms will first be validated in a digital twin environment that mirrors real CPS operations. This simulation-based approach enables rigorous testing of adaptation strategies under realistic conditions prior to field deployment. The framework will also be optimized for edge execution, supporting low-latency, localized anomaly detection even in offline or bandwidth-constrained settings.

Another key objective is to eliminate the dependency on labeled fault data in second stage. To this end, we plan to explore semi-supervised and self-supervised learning techniques such as clustering and contrastive learning to differentiate between fault-like and attack-like anomalies without requiring explicit fault annotations. This will improve the framework's applicability in real-world deployments where labeled fault data is limited, unavailable, or costly to obtain.

We also envision integrating autonomous AI agents to manage detection and system-level adaptation in real time. These agents can monitor local detection performance, dynamically ad-

just parameters, and autonomously trigger retraining or data augmentation when degradation is detected. In distributed CPS environments such as smart grids with geographically dispersed nodes agents may collaborate via shared communication infrastructure or distributed control platforms. For instance, anomaly signals and learned patterns can be exchanged through publish-subscribe networks or edge-cloud systems, enabling coordinated responses such as localized containment, load balancing, or predictive maintenance. Together, these enhancements including adaptive learning, label-efficient learning, and decentralized agent-based coordination will advance the proposed framework into a robust, scalable, and adaptive solution for real-time industrial CPS deployments.

References

[1] A. Humayed, J. Lin, F. Li, and B. Luo, "Cyber-physical systems security—a survey," *IEEE Internet of Things Journal*, vol. 4, no. 6, pp. 1802–1831, 2017.

[2] A. A. Cárdenas, S. Amin, and S. Sastry, "Research challenges for the security of control systems," *HotSec*, vol. 8, pp. 6–6, 2008.

[3] W. L. Duo, M. C. Zhou, and A. Abusorrah, "A survey of cyber attacks on cyber physical systems: Recent advances and challenges," *IEEE/CAA Journal of Automatica Sinica*, vol. 9, no. 5, pp. 784–800, 2022.

[4] Cybersecurity and Infrastructure Security Agency, "Cyberattacks on critical infrastructure," 2023. [Online]. Available: https://www.cisa.gov/topics/critical-infrastructure-security-and-resilience

[5] S. Mubarak, M. Habaebi, M. Islam, F. Rahman, and M. Tahir, "Anomaly detection in ics datasets with machine learning algorithms," *Computer Systems Science and Engineering*, vol. 37, no. 1, pp. 33–46, 2021.

[6] E. Anthi, L. Williams, P. Burnap, and K. Jones, "A three-tiered intrusion detection system for industrial control systems," *Journal of Cybersecurity*, vol. 7, 2021.

[7] Y. Harada, Y. Yamagata, O. Mizuno, and E. H. Choi, "L-based anomaly detection of cps using a statistical method," in *2017 8th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, 2017, pp. 1–6.

[8] S. Adepu, T. Semwal, and A. Mathur, "A modular testbed for realistic industrial control system cybersecurity analysis," *arXiv preprint arXiv:2210.13325*, 2022. [Online]. Available: https://arxiv.org/abs/2210.13325

[9] Sandia National Laboratories, "Emulation and detection of physical faults and cyber-attacks on building energy systems: Demonstrating the cyber-physical emulation engine for buildings (cpeb)."

[10] University of Illinois, “Co-simulation platform for cps,” 2023. [Online]. Available: https://publish.illinois.edu/cps-testbed/files/2023/06/CPS-Testbed-Poster-1.pdf

[11] W. Hao, T. Yang, and Q. Yang, “Hybrid statistical-machine learning for real-time anomaly detection in industrial cyber—physical systems,” *IEEE Transactions on Automation Science and Engineering*, vol. 20, no. 1, pp. 32–46, 2023.

[12] M. Syfert, A. Ordys, J. Kościelny, P. Wnuk, J. Możaryn, and K. Kukiełka, “Integrated approach to diagnostics of failures and cyber-attacks in industrial control systems,” *Energies*, vol. 15, p. 6212, 2022.

[13] A. Dehlaghi-Ghadim, A. Balador, M. H. Moghadam, H. Hansson, and M. Conti, “Icssim —a framework for building industrial control systems security testbeds,” *Computers in Industry*, vol. 148, p. 103906, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166361523000568

[14] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimensional distribution,” *Neural Computation*, vol. 13, no. 7, pp. 1443–1471, 2001.

[15] D. M. J. Tax and R. P. W. Duin, “Support vector data description,” *Machine Learning*, vol. 54, no. 1, pp. 45–66, 2004.

[16] Q. Xu, S. Ali, and T. Yue, “Digital twin-based anomaly detection in cyber-physical systems,” pp. 205–216, 2021. [Online]. Available: https://doi.org/10.1109/ICST49551.2021.00031

[17] X. Zhou, W. Liang, S. Shimizu, J. Ma, and Q. Jin, “Siamese neural network based few-shot learning for anomaly detection in industrial cyber-physical systems,” *IEEE Transactions on Industrial Informatics*, vol. 17, no. 8, pp. 5790–5798, 2021. [Online]. Available: https://doi.org/10.1109/TII.2020.3047675

[18] N. Jeffrey, Q. Tan, and J. R. Villar, “Using ensemble learning for anomaly detection in cyber—physical systems,” *Electronics*, vol. 13, no. 7, p. 1391, 2023.

[19] Y. Du, Y. Huang, G. Wan, and P. He, “Deep learning-based cyber-physical feature fusion for anomaly detection in industrial control systems,” *Mathematics*, vol. 10, no. 22, p. 4373, 2022. [Online]. Available: https://doi.org/10.3390/math10224373

[20] N. Jeffrey, Q. Tan, and J. R. Villar, “A hybrid methodology for anomaly detection in cyber—physical systems,” *Neurocomputing*, vol. 568, p. 127068, 2024.

[21] E. Bartocci, N. Manjunath, L. Mariani, C. Mateis, and D. Ničković, “Cpsdebug: Automatic failure explanation in cps models,” *International Journal on Software Tools for Technology Transfer*, vol. 23, no. 5, pp. 783–796, 2021. [Online]. Available: https://doi.org/10.1007/s10009-020-00599-4

[22] J. Boi-Ukeme, C. Ruiz-Martin, and G. Wainer, “Real-time fault detection and diagnosis of cps faults in devs,” in *2020 IEEE 6th International Conference on Dependability in Sensor, Cloud and Big Data Systems and Application (DependSys)*. IEEE, 2020, pp. 57–64. [Online]. Available: https://doi.org/10.1109/DependSys51298.2020.00017

[23] H. Ruan, B. Dorneanu, H. Arellano-Garcia, P. Xiao, and L. Zhang, “Deep learning-based fault prediction in wireless sensor network embedded cyber-physical systems for industrial processes,” *IEEE Access*, vol. 10, pp. 10 867–10 879, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3144333

[24] K. Zhang, C. Keliris, T. Parisini, and M. M. Polycarpou, “Identification of sensor replay attacks and physical faults for cyber-physical systems,” *IEEE Control Systems Letters*, vol. 6, pp. 1178–1183, 2022.

[25] M. Taheri, K. Khorasani, I. Shames, and N. Meskin, “Cyberattack and machine-induced fault detection and isolation methodologies for cyber-physical systems,” *IEEE Transactions on Control Systems Technology*, vol. 32, no. 2, pp. 502–517, 2024.

[26] D. Kim, S. C. Han, Y. Lin, B. H. Kang, and S. Lee, “Rdr-based knowledge based system to the failure detection in industrial cyber physical systems,” *Knowledge-Based Systems*, vol. 150, pp. 1–13, 2018. [Online]. Available: https://doi.org/10.1016/j.knosys.2018.02.009

[27] “Wireshark: a network traffic analyzer,” https://github.com/wireshark/wireshark, accessed: 2024-11-02.

[28] “Tcpdump: The classic command-line packet analyzer,” https://www.tcpdump.org/, accessed: 2025-04-26.

[29] “A rocket-fast system for log processing,” https://github.com/rsyslog/rsyslog, accessed: 2024-11-05.

[30] “Docker container manual,” https://docs.docker.com/manuals/, accessed: 2024-12-05.

[31] “Scikit-learn: machine learning in python,” https://github.com/scikit-learn/scikit-learn, accessed: 2025-03-05.

[32] “Tensorflow - an open source machine learning framework for everyone,” https://github.com/tensorflow/tensorflow, accessed: 2024-10-16.

[33] “Matplotlib: plotting with python,” https://github.com/matplotlib/matplotlib, accessed: 2025-03-05.

[34] “Pandas: powerful python data analysis toolkit,” https://github.com/pandas-dev/pandas, accessed: 2025-03-05.

[35] “Statistical data visualization in python,” https://github.com/mwaskom/seaborn, accessed: 2025-03-05.

[36] S. Bird, E. Loper, and E. Klein, *Natural Language Processing with Python*. O’Reilly Media Inc., 2009.

[37] “Optuna: A hyperparameter optimization framework,” https://github.com/optuna/optuna, accessed: 2025-03-05.

[38] A. Dehlaghi-Ghadim, M. H. Moghadam, A. Balador, and H. Hansson, “Anomaly detection dataset for industrial control systems,” 2023. [Online]. Available: https://arxiv.org/abs/2305.09678