

DESIGN AND DEVELOPMENT OF AI BASED INTRUSION DETECTION AND MONITORING DASHBOARD FOR PID- CONTROLLED SYSTEMS

Malik Zawar Ahmed

University of Bedfordshire

malikzawar63@gmail.com

DOI: <https://doi.org/10.5281/zenodo.22892096>

Keywords

Artificial Intelligence, Intrusion Detection System, Industrial Control Systems, PID Controller, Cyber-Physical Systems, Machine Learning, Anomaly Detection, Industrial IoT (IIoT)

Article History

Received: 01 June 2026

Accepted: 23 August 2026

Published: 08 September 2026

Copyright @Author

Corresponding Author: *

Malik Zawar Ahmed

Abstract

This paper describes the design and development of an AI- based intrusion detection and monitoring system for PID- controlled industrial environments. The project addresses the increasing issues of cybersecurity in Industrial Control Systems (ICS) as a result of the convergence of operational technologies and networked systems. We constructed a simulated cyber-physical environment, which produces real-time operational data from PID-controlled processes like temperature and water-level regulating. The system includes attack simulation techniques such as replay attacks, noise injection and false data injection (FDI) attacks to analyse the industrial processes security vulnerabilities.

Machine Learning methods such as Random Forest and Logistic Regression are used for real time cyber intrusion detection and anomaly detection. Finally, the reliability and efficacy of the detection performance are evaluated by accuracy, precision, recall, F1-score and confusion matrix analysis. A web-based dashboard is combined with a real-time detection engine providing live monitoring, visual analytics, alarms and logging of historical events to improve situational awareness.

The suggested system makes a contribution in the field of industrial cybersecurity by combining artificial intelligence, intrusion detection, and PID control system dynamics into a single framework. This research presents a scalable and effective strategy for safeguarding current ICS and Industrial IoT infrastructures through adaptive real-time anomaly detection inside a simulated industrial setting, contrasting with classic static IDS methods.

CHAPTER 1: INTRODUCTION

Industrial Control Systems (ICS) are a vital part of the key infrastructure sectors like manufacturing, electricity generation, water treatment, transportation and chemical processing. Such systems rely heavily on automatic control systems,

in particular Proportional-Integral-Derivative (PID) controllers, to provide steady and productive industrial processes. In recent years, industrial systems have become much more exposed to cyber-attacks due to the convergence of operational technology with information technology and Industrial Internet of Things (IIoT) environments.

High-profile cyber disasters such as Stuxnet, Ukrainian Power Grid assault and TRITON malware have shown that cyber-attacks on industrial processes can cause serious operational disruption, financial loss and physical harm.

Traditional intrusion detection systems mainly focus on network traffic analysis and signature-based detection approaches. However, such tactics are generally ineffectual in industrial environments as attacks against ICS tend to affect process behavior instead of traditional IT assets. In addition, current systems struggle to detect complex attacks like replay attacks, fake data injections (FDI), and noise injection attacks, particularly when these attacks resemble regular operational behavior. This, in turn, creates a demand for intelligent, adaptive and process-aware cybersecurity systems that can detect anomalies in real time.

The project aims to develop an AI-Based Intrusion Detection and Monitoring Dashboard for PID-Controlled Systems. The project integrates machine learning approaches, cyber-physical systems modelling and real-time monitoring technologies in a unified framework for industrial cybersecurity. An industrial process simulation driven by a PID-controller is designed to generate operational data for normal and attack situations. We inject a number of cyber-attack scenarios into the simulated environment to generate labelled datasets for training and evaluating machine learning. Supervised machine learning methods such as Logistic Regression, Decision Tree and Random Forest techniques are implemented for detecting anomalous process behavior and effectively classify attack types.

The suggested system further incorporates a real-time detection engine embedded with a Flask-based online dashboard that performs live process visualization, alert generation, logging of past events, and threat monitoring capabilities. Unlike

many typical IDS solutions based on offline analysis, this system is able to continuously monitor and adaptively detect intrusions in a simulated industrial setting. The research makes a contribution to the field of industrial cybersecurity by bridging the gap between control systems engineering and AI-driven security analytics. Moreover, the modular architecture and the lightweight technology stack allow the scalability of the system and suitability for the future implementation in the Industrial IoT contexts.

This report is divided into a number of chapters. Chapter 1 describes the project aims, objectives and the artefact description. Chapter 2 presents an overview of literature pertaining to industrial cybersecurity, intrusion detection systems, machine learning and cyber physical systems. Chapter 3 presents research methodology, system architecture and design strategy. Chapter 4 describes implementation of the PID simulator, attack modules, machine learning pipeline and dashboard components. Chapter 5 assesses the performance of the system using cybersecurity and machine learning metrics. Finally, chapter 6 ends the project, evaluates limitations and gives recommendations for future improvements.

1.1 Goals

The main objective of this project is to create, implement and evaluate an artificial intelligence-based intrusion detection and monitoring system that is capable of identifying in real time cyber-attacks against PID-controlled industrial processes. The project aims at improving the security and resilience of Industrial Control Systems by merging machine learning-based anomaly detection and process-aware monitoring.

The system aims to mimic genuine industrial contexts where PID controllers are used for controlling process variables like temperature and water level. The suggested system aims to detect

cyber-attacks, which may be missed by typical signature-based security mechanisms, by analysing operational data and finding anomalous behavioural patterns. The project will also boost situational awareness with an interactive web dashboard that delivers live visualisation, operational insights and security alerts for system managers and operators.

Another key objective is to show how artificial intelligence approaches may be successfully used in industrial cybersecurity situations without the need for expensive infrastructure and/or unique hardware. The project thus aims to build a lightweight, scalable and flexible solution using accessible technologies such as Python, Scikit-Learn, Flask, SQLite and Chart.js.

1.2 Objectives

The aims of this initiative are:

- To design and execute a realistic simulation of PID controller, which is capable of modelling industrial processes such as temperature regulation and water-level control.
- Design an attack simulation module that can launch replay attacks, noise injection attacks and false data injection (FDI) attacks in the simulated industrial environment.
- To produce labelled operational datasets containing normal and malicious process activity for training and evaluation of machine learning.
- Use feature engineering to extract significant temporal and statistical features from PID process data to improve anomaly detection performance.
- Design, train and assess supervised machine-learning models including Logistic Regression, Decision Tree and Random Forest classifiers for the purpose of intrusion detection.
- To evaluate the models' performance based on cybersecurity-specific measures, including accuracy, precision, recall, F1-score, confusion matrix analysis, and false negative rates.

- To create a real-time intrusion detection engine that can process real-time PID readings and produce alerts with a minimal detection latency.
- Development & deployment of a Flask based web dashboard for real time monitoring, visual analytics, alert management and historical logging capability.
- To verify the reliability, accuracy and efficacy of the proposed system through unit testing, integration testing and system level testing.
- To design a modular and scalable architecture, which can be further improved in the future and perhaps deployed in Industrial Internet of Things (IIoT) scenarios.

1.3 Research Challenges

The artifact developed in this project is an AI based intrusion detection and monitoring system intended for PID controlled industrial contexts. The solution combines process modelling, cyber-attack modelling, machine learning-based anomaly detection and real-time visual monitoring into a unified operational framework.

The item features a discrete-time PID process simulator, written in Python. The simulator emulates genuine industrial process behaviour and continuously generates operational data such as setpoint values, process outputs, controller outputs and error measurements. An attack simulation module injects numerous types of cyber-attacks into the process data, such as replay attacks, false data injection attacks and noise injection attacks. These attacks are designed to mimic actual world cyber-physical risks found in industrial environments.

The generated datasets are processed through a machine learning pipeline consisting of feature engineering, data normalisation, model training and classification. Several supervised learning algorithms are assessed and the best performing one is put to the real-time detection engine. The

detection engine continuously evaluates incoming process readings, recognizes aberrant behaviour, classifies attack types and records all operational data and alarms in a SQLite database.

The artefact also has a web-based monitoring dashboard made using Flask, HTML, CSS, JavaScript and Chart.js. The dashboard offers live process visualization, real-time threat detection indications, alert feeds, attack simulation controls, historical logs, and CSV export options. The system is designed with a modular architecture with simulation, machine learning, detection, database and dashboard layers to allow future extensibility and connection with more advanced industrial cybersecurity solutions.

CHAPTER 2:

Literature Review

2.1 Introduction

The integration of Information Technology (IT) and Operational Technology (OT) has changed Industrial Control Systems (ICS) into more linked and efficient systems but also increased their vulnerability to complex cyber threats. As the adoption of the Industrial Internet of Things (IIoT) paradigm spreads to industries throughout the world, securing vital infrastructure components, such as systems controlled by Proportional-Integral-Derivative (PID) controllers, has become an increasingly important topic of research. The repercussions of successful cyber-attacks on such systems may extend beyond data loss, including physical damage, service disruption and even loss of human life (Kayan et al., 2022).

This chapter provides a critical review of the relevant literature that underpins the main research areas of this project: the cybersecurity landscape of ICS, PID control systems and cyber-physical system modeling, attack taxonomies applicable to the industrial environment, traditional and machine learning based intrusion

detection approaches, and real-time monitoring solutions. This review seeks to systematically analyze the strengths and limitations of current research and to identify the gaps that the proposed system addresses. The literature was mainly located in peer-reviewed academic journal and conference proceedings published in the recent five years, with the exception of key older works cited for context.

2.2 Industrial Control Systems and Cybersecurity Landscape

Industrial control systems are made up of different types of control mechanisms utilized in critical infrastructure including Supervisory Control and Data Acquisition (SCADA) systems, Distributed Control Systems (DCS), and Programmable Logic Controllers (PLCs). A detailed analysis of cybersecurity issues for ICS environments and pointed out that the transition from isolated, air-gapped networks to IP-connected architectures has created a wide attack surface that was not foreseen in the original design of these systems (Kayan et al., 2022). The authors argue that the security-by-obscurity concept, historically used in ICS, is no longer sufficient in current linked environments. These vulnerabilities are serious and real-world situations have shown this to be the case. The well-known Stuxnet malware that attacked the PLC controllers of the Iranian nuclear facilities' centrifuges and caused physical damage by quietly altering process setpoints (Langner, 2011). More recently, the TRITON/TRISIS malware was developed to target Schneider Electric's Safety Instrumented Systems (SIS), demonstrating that adversaries are now explicitly targeting safety-critical process control equipment (Dragos, 2017). The attacks on the Ukrainian Power Grid in 2015 and 2016, attributed to the Sandworm threat group, resulted in widespread power outages affecting hundreds of thousands of civilians and

demonstrated the real-world implications of nation-state attacks on industrial infrastructure (Lee, Assante and Conway, 2016).

Risk assessment methodology for the SCADA environment where the authors argue that traditional risk models from IT are inadequate to address the safety and availability needs of OT systems (Cherdantseva et al., 2016). Their analysis points to the lack of real-time monitoring and process-aware threat identification as key holes in current ICS security. In a similar context, the security requirements for IIoT systems and highlighted that the threat landscape is changing rapidly and adaptive detection mechanisms are needed to cope with new attack patterns without prior knowledge of signatures as legacy ICS components are gradually connected to cloud platforms and remote access interfaces (Alcaraz and Zeadally, 2023).

2.3 PID Controllers and Modeling of Cyber Physical Systems

The Proportional-Integral-Derivative (PID) controller is one of the most extensively used feedback control devices in industrial automation. An extensive review of PID tuning methods and their application to energy systems mentioned that PID controllers are ubiquitous in processes such as temperature regulation, flow control, pressure management, and water-level control precisely because of their simplicity, robustness, and well-understood mathematical behaviour (Li, 2023). The distinctive response of a PID controlled process, including setpoint monitoring, error correction and steady state stability, provides a rich supply of behavioural data that can be leveraged for anomaly identification.

Cyber-Physical Systems (CPS) link cyber components and physical components together into one system and cyberattacks can affect physical consequences. The security vulnerabilities

of CPS and found that assaults targeting the sensing and actuation layers, which directly interact with PID control loops, are the most hazardous threat category as their impacts directly propagate into the physical world (Humayed et al., 2017). Based on this, A mathematical framework to characterize the vulnerability of closed-loop PID controlled systems to false data injection attacks, analytically demonstrating that even small, carefully crafted perturbations on sensor readings can destabilize an otherwise stable process without triggering conventional alarm thresholds (Ding et al., 2021). This analysis is very important to the current project, as it highlights how process aware anomaly detection is a necessity, rather than a basic threshold-based alerting.

Simulation-based approaches to generate ICS security datasets have gained appeal in the research community, given the ethical and practical constraints to experimentation on real industrial infrastructure. The SWaT (Secure Water Treatment) and BATADAL benchmark datasets through hardware-in-the-loop simulation of water treatment processes that have been widely used as standard references for the evaluation of ICS intrusion detection methods (Ahmed et al., 2017). Their methodology for simulating regular operations and targeted attacks in a controlled environment is quite similar to the methodology of the current study. Therefore, they validate the use of simulation-based dataset creation for ICS cybersecurity research and demonstrate its reproducibility.

2.4 Industrial Environment Cyber Attack Taxonomies

Understanding the nature and features of attacks on ICS environments is a prerequisite for creating efficient detection techniques. There are various attack categories identified in the literature, which are especially significant for PID-controlled

systems. False Data Injection (FDI) attacks are the modification of the sensor measurements to mislead the control system to make wrong actuation decisions. Mokhtari and Yen (2024) investigated FDI assaults on load frequency control systems and found that adversaries can generate stealthy injections inside the typical sensor variance limits, rendering them undetectable by conventional anomaly thresholds. The detection approach relied on machine learning and was highly precise, but required precisely annotated training data, a constraint that is tackled in the current study by generating structured datasets.

Another major category of assaults in ICS is the use of replay attacks, where an adversary captures and retransmits previously recorded lawful sensor data to hide a continuing physical manipulation of the process. A hybrid passive-active detection methodology for the FDI and replay threats in the industrial control systems (Ma and Tsou, 2024). They employed both passive monitoring of sensor statistics and active perturbation signals to expose inconsistencies introduced by replayed data. The main advantage of their system was that it was able to detect replay attacks without prior knowledge of the timing of the attack. The active perturbation component added complexity to their system, which may not be desirable in safety critical contexts.

Noise injection attacks, also known as Gaussian noise or random perturbation attacks, insert stochastic perturbations into the data streams of sensors to degrade control performance or conceal anomalous behaviour in statistical noise. Computationally efficient algorithms for cyberattack detection in ICS and included noise injection as one of the attack types they considered, which they noted is difficult to detect using single-feature anomaly detectors as it resembles natural measurement noise (Marusak et

al., 2024). Their study highlights the necessity of multi-feature, multivariate detection techniques – a premise that precisely underpins the feature engineering workflow utilized in this project.

2.5 Traditional IDS and Their Limitations in ICS

Intrusion detection systems are traditionally categorized into two basic types, signature-based and anomaly-based detection. Signature-based systems such as Snort and Suricata compare network traffic or system events against a database of known attack patterns. These solutions are essentially reactive, needing knowledge of attack signatures in advance and completely failing against novel or zero-day incursions (Tama, Lee and Lee, 2022), though successful against well-documented threats. In ICS environments, where attack vectors generally rely on manipulation of process variables rather than traditional network payloads, signature-based detection is especially poor.

Statistical anomaly detection techniques such as control charts, and CUSUM (Cumulative Sum) algorithms have been used in the context of process monitoring for finding deviations from expected operational parameters. The statistical process control methods to detect FDI attacks in networked control systems (Deng et al., 2016). They found that the CUSUM-based methods have low computational overhead, but they have a hard time to detect stealthy, low-amplitude attacks that fall within the statistical thresholds. In addition, univariate approaches fail to account for the inter-variable correlations that are typical of multi-loop PID systems and, therefore, cannot identify concerted attacks on many process variables concurrently.

The literature also examined the specification-based IDS techniques for ICS. These systems specify explicit models of acceptable system

behaviour and generate warnings when observed activity diverges from the specification. Specification-based methods can be highly accurate for well-defined industrial processes but require a lot of domain knowledge to construct complete behaviour models and are not able to adapt to changes in process operating modes without manually updating the models (Kayan et al., 2022). These restrictions together provide a strong incentive to embrace data-driven, machine learning systems, which can learn normal behaviour straight from process data.

2.6 Machine Learning-Based Intrusion Detection in ICS

The use of machine learning in intrusion detection in ICS has become a very active research direction. A thorough mapping research and empirical comparison of data-driven intrusion detection algorithms in industrial control networks (Tama, Lee and Lee, 2022). A total of 27 different approaches were evaluated on many benchmark datasets. They observed that ensemble approaches, and Random Forest in particular, always outperformed single classifiers such as Support Vector Machines (SVM) and k-Nearest Neighbours (kNN) in terms of F1-scores and false negative rates. The authors also pointed out that the detection performance was more affected by the choice of feature representation than the classifier choice, thus underlining the crucial role of feature engineering in the design of ML-based IDS.

Within the literature on ICS security, Random Forest classifiers have been particularly interesting. A machine learning-based framework to detect and mitigate sensor fake data injection attacks in industrial control systems in real-time (Elnour et al., 2023). In this case, Random Forest is used as the primary classifier and a model-based state estimator is used to cross-validate the detection.

Their approach achieved above 97% detection accuracies for the SWaT dataset, while the Random Forest component played a major role in reducing false positive rates. The study also added a mitigation layer that replaced predicted accurate sensor values if an assault was detected, a feature that moved the framework from passive detection to active response. However, the complexity and possible impracticality of their approach in resource-constrained ICS deployments arise from the use of a high-fidelity physics model for state estimation.

Several ICS intrusion detection studies have assessed Logistic Regression as a baseline, and as part of ensemble approaches, due to its relative simplicity as a classifier. Mokhtari and Yen (2024) extended their comparative evaluation to include Logistic Regression and found it a good choice for the detection of attack scenarios with large distributional shifts, but it was less successful for FDI attacks with smaller distributional shifts, where the decision boundary in feature space was non-linear. The advantage of Decision Trees is that they are naturally interpretable, which is very useful in operational ICS situations where security analysts need to comprehend and justify warnings. However, decision trees are prone to overfitting to the training data and often have inferior generalisation performance than ensemble variations such as Random Forest (Tama, Lee and Lee, 2022).

Deep learning techniques such as Long Short-Term Memory (LSTM) and Convolutional Neural Network (CNN) are also studied for anomaly identification in ICS. LSTM-based autoencoders can detect anomalies in the SWaT dataset with good accuracy, by learning temporal correlations in multi-sensor time-series data (Kravchik and Shabtai, 2021). Their approach obtained good detection rates without requiring tagged attack data during training, which may be a potential

advantage in contexts where attack samples are sparse. However, the significantly increased computing costs and decreased interpretability of deep learning models pose practical obstacles for deployment in real-time, resource-constrained ICS monitoring applications. These restrictions motivate the use of the computationally efficient supervised classifiers (Logistic Regression, Decision Tree and Random Forest) as the main detection techniques in the current research.

2.7 Feature Engineering of Process Data for Anomaly Detection

The need of effective feature engineering in the development of high-performance machine learning models for ICS security is well understood. The raw PID process data (setpoint, process variable, error, and controller output readings) to be converted in order to encode statistical and temporal features of normal and abnormal operation. Utilizing the SWaT dataset, the computed features such as rolling means, standard deviations, and rate-of-change metrics substantially enhanced the sensitivity of anomaly detection models compared to employing only the raw sensor inputs (Goh et al., 2017).

Domain-specific features obtained from a control-theoretic analysis can enhance discriminative power for systems managed by PID. The use of features based on integral absolute error (IAE) and settling time estimates for supervised classification, stating that these metrics better capture the cumulative effect of attacks on process performance than immediate measures (Deng et al., 2016). Marusak et al. (2024) also discovered that adding lagged sensor data and cross-variable correlation characteristics significantly increased the detection rate of coordinated multi-variable attacks. These results directly feed the feature engineering method applied in the current project, including rolling statistics, error

derivatives and inter-signal correlation measures in the classification pipeline.

2.8 Real-Time Monitoring and Dashboards for ICS Security

The operational success of an intrusion detection system depends not only on the detection algorithm but also on how alarms and situational awareness are relayed to security operators. Commercial ICS monitoring solutions like Claroty, Nozomi Networks, and Dragos Platform offer wide network-level visibility and threat intelligence but are largely network-traffic focused and do not have deep integration with process-level data from individual PID control loops (Alcaraz and Zeadally, 2023). Also, because to their commercial nature and high cost, they are not accessible for research prototyping and evaluation.

In the research community, web-based dashboard frameworks have been widely used to construct specialized security monitoring interfaces. A lightweight ICS security monitoring prototype with Flask and WebSocket technologies, which visualizes network traffic anomalies in real-time with sub-second update latency was built by (Ullah and Mahmoud, 2020). They showed that it was possible to create responsive browser-based security dashboards without hefty SCADA HMI software. The use of Chart.js and other JavaScript visualization libraries within Flask applications has become a popular approach to the display of real-time time-series data, especially for projects that are aimed for educational or research deployments with low scalability requirements.

Several academics have stressed the need of security dashboards with alert management and historical event logging functionalities. A user study with industrial operators and found that dashboards that provide too many notifications without contextual prioritization led to alert fatigue and operators dismissing actual security

events (Hadžiosmanović et al., 2014). This conclusion highlights the need to include intelligent alarm filtering, severity categorization and historical trend visualization inside the monitoring interface – all of which are covered in the dashboard component of the current project.

2.9 Research Gap and Motivation for Proposed System

An overview of the available literature shows certain critical gaps that inspire the present investigation. First, while machine learning-based IDS solutions for ICS have been shown to be effective in detecting anomalies in published benchmarks, the majority of these studies use pre-existing datasets such as SWaT or BATADAL for evaluation. However, these datasets represent specific process topologies and attack scenarios that may not be applicable to other industrial environments. Research is required in order to illustrate how labelled datasets may be generated from arbitrary PID driven processes to allow for the generation of environment specific training data for tailored intrusion detection systems.

Second, most of the published research on intrusion detection in ICSs concerns the detection

algorithm in isolation, and reports offline evaluation findings, without proving integration into a full-fledged, operational monitoring pipeline. Relatively little attention has been paid to the gap between an evaluated classifier and a deployable real-time detection system, including data ingestion, feature calculation, model inference, warning production, and operator interface. This research directly solves this gap by building an end-to-end system that incorporates simulation, detection and monitoring into a unified operational framework.

Third, the literature suggests that there are few lightweight open-source ICS security monitoring prototypes available, which can serve as accessible platforms for security teaching, research evaluation, and proof-of-concept creation in academia. The suggested system utilizes freely available Python-based technology and seeks to contribute to this area by offering a modular, extensible platform which can be easily applied to different types of industrial processes and extended with additional detection capabilities in future work.

2.10 Comparison of Traditional Systems vs Proposed System

Table 1 Comparison of Traditional Systems vs Proposed System

Feature of Traditional IDS Systems	Limitations	Proposed AI-Based IDS for PID-Controlled Systems
1. Detection Method	Signature-based and rule-based detection Tama et al. (2022); Pinto et al. (2023)	Machine Learning-based anomaly detection using Random Forest, Decision Tree, and Logistic Regression
2. Awareness of Industrial Processes	Primarily network-focused; limited understanding of physical processes Kayan et al. (2022); Koay et al. (2023)	Process-aware detection using PID controller behavior and operational parameters
3. Detection of Unknown Attacks	Poor performance against zero-day and unknown attacks Momand et al. (2023); Pinto et al. (2023)	Capable of identifying previously unseen anomalies through learned behavior patterns

4. False Data Injection (FDI) Attack Detection	Difficult to detect stealthy FDI attacks Mokhtari and Yen (2024); Elnour et al. (2023)	Specifically trained to detect FDI attacks using labeled datasets
5. Replay Attack Detection	Limited capability Ma & Tsou (2024); Marusak et al. (2024)	Designed to identify replay attacks through process behavior analysis
6. Noise Injection Attack Detection	Often misclassified as normal sensor noise Marusak et al. (2024); Koay et al. (2023)	Uses multi-feature analysis to distinguish malicious noise from normal variations
7. Real-Time Monitoring	Often relies on offline analysis and log review Elnour et al. (2023)	Continuous real-time monitoring and intrusion detection
8. Adaptability	Requires manual signature updates Tama et al. (2022); Momand et al. (2023)	Learns from operational data and adapts to changing environments
9. Feature Analysis	Uses packet signatures and predefined rules Tama et al. (2022); Koay et al. (2023)	Uses process variables, control errors, rolling statistics, and temporal features
10. Detection Accuracy	Lower accuracy for process-level attacks Elnour et al. (2023); Pinto et al. (2023)	Higher detection accuracy through machine learning classification
11. Alert Generation	Basic alerts based on predefined rules (Holdbrook et al., 2024)	Intelligent real-time alerts based on anomaly classification
12. Visualization Support	Limited or absent Ullah and Mahmoud (2022)	Interactive Flask dashboard with live charts and visual analytics
13. Historical Event Logging	Basic logging mechanisms (Holdbrook et al., 2024)	Comprehensive event storage, threat history, and export functionality
14. Scalability	Difficult to adapt to new industrial environments Koay et al. (2023); Alcaraz and Zeadally (2023)	Modular and scalable architecture suitable for IIoT expansion
15. Deployment Cost	Commercial solutions can be expensive Ullah and Mahmoud (2022); Koay et al. (2023)	Built using open-source technologies (Python, Flask, Scikit-Learn, SQLite)
16. Process Simulation Support	No integrated simulation capability Ahmed et al. (2017); Koay et al. (2023)	Includes PID process simulator and attack simulation environment
17. Dataset Generation	Depends on external datasets and signatures Dehlaghi-Ghadim et al. (2023)	Generates custom labeled datasets for specific industrial scenarios
18. Research and Educational Use	Limited access due to proprietary solutions Koay et al. (2023); Pinto et al. (2023)	Open, lightweight, and suitable for research and academic purposes
19. System Architecture	Detection component often operates independently Elnour et al. (2023)	End-to-end integrated framework including simulation, ML, detection, database, and dashboard

20. Industrial IoT Readiness	Limited support for modern IIoT environments Alcaraz and Zeadally (2013); Koay et al. (2023)	Designed for future deployment in Industrial IoT (IIoT) infrastructures
------------------------------	--	---

2.11 Summary

The state of the art in five areas—ICS cybersecurity, PID control systems and cyber-physical system modelling, industrial attack taxonomies, machine learning-based intrusion detection, and real-time monitoring solutions—that are crucial to the suggested research have been examined in this chapter. The literature demonstrates that the sophisticated, process-aware attacks that pose a danger to contemporary PID-controlled industrial environments cannot be detected by conventional signature-based and statistical intrusion detection systems. In previously published research, machine learning techniques—in particular, ensemble classifiers like Random Forest—have shown higher detection performance, underscoring the significance of efficient feature engineering and labelled dataset development. Literature clearly lacks end-to-end, lightweight, open-source solutions that combine real-time monitoring, ML-based detection, and process simulation into a single operational framework. By building on the methodological assumptions set by the reviewed literature and offering a fresh, integrated approach to industrial cybersecurity, the suggested methodology directly tackles these highlighted shortcomings.

CHAPTER 3:

Research Methodology

3.1 Introduction

This chapter discusses the research approach used in design, development and evaluation of AI Based Intrusion Detection and Monitoring System for PID Controlled Industrial Environment. In this chapter we focus on the 'how', i.e. the strategies, methods, tools and

procedures utilized to carry out the inquiry and build the proposed system, rather than repeating what was found or produced. Each methodological choice is justified in terms of the nature of the problem; the restrictions of the project environment and the precedents set in the examined literature.

The chapter is organized as follows: research philosophy and design strategy, development methodology, system architecture overview, dataset generation strategy, attack simulation methodology, machine learning pipeline design, feature engineering approach, evaluation strategy, tools and technologies selection, and project management. These parts together give a full description of the research design and implementation, from initiation through to evaluation.

3.2 Research Philosophy and Design Strategy

This project adheres to a Design Science Research (DSR) paradigm, which is particularly suitable for applied computing projects aimed primarily at the design and evaluation of an artefact to solve a well-defined practical problem (Hevner et al., 2004). The outputs of DSR are both prescriptive, indicating how a system should be designed, and evaluative, judging if the artefact built has achieved its intended aims, which sets it apart from solely empirical or theoretical research. The main contribution of this study is an operational end to end intrusion detection system and not a theoretical model. Therefore, DSR is the best suitable epistemological framework.

The research under the DSR framework is based on a constructive approach rooted on a pragmatic philosophy. The pragmatist position asserts that

the worth of a research method or theoretical model is based on its practical utility and its capacity to address real-world problems (Creswell and Creswell, 2018). This is relevant here because the project is directly inspired by the practical inadequacies of existing intrusion detection systems in industrial situations, as proven in literature study. Pragmatism also supports the mixed-methods orientation of the research, integrating quantitative performance evaluation of machine learning classifiers with system-level functional assessment of the integrated detection and monitoring pipeline.

Simulation-based experimental strategy is the major tool for data creation and system evaluation. This option is supported on ethical and practical grounds: direct testing on active industrial infrastructure is dangerous, legally restricted and operationally infeasible inside an academic enterprise. Simulation-based research has become a popular choice in the ICS security community (e.g., the SWaT and BATADAL benchmark datasets, and is considered a valid and reproducible method for generating labelled

cybersecurity datasets when real-world data collection is not possible (Ahmed et al., 2017).

3.3 Development Methodology

The project was developed utilizing an Agile methodology and was organized in two-weeks structured sprints. Agile was preferred over sequential (Waterfall) methods for a variety of reasons pertinent to the context of the project. The five major components of the system – the PID simulator, attack injection module, machine learning pipeline, real-time detection engine, and web dashboard – are interdependent, meaning that the requirements for later components could only be precisely defined once earlier components had been prototyped and validated. Agile offers this incremental refinement, but Waterfall has to have the requirements fully specified before development can begin. Second, the machine learning part included an inherent uncertainty: model selection, hyperparameter tuning and feature engineering decisions were based on empirical outcomes of earlier trials; thus, a fixed upfront design was not appropriate. The project was broken into five major Agile sprints.

Processes	Sprint 1	Sprint2	Sprint 3	Sprint 4	Sprint 5
Literature Review and System Architecture design					
PID Process stimulator and Attack injection Module development					
Dataset Generation, feature engineering and Evaluation of Machine learning model					
Development of real time detection engine, SQLite database and dashboard Prototype					
Integration Testing, System level Evaluation and Documentation					

Figure 1 Gantt Chart of Agile Sprints

Throughout the project a task management methodology was followed, keeping a product backlog, and at the end of each sprint cycle a retrospective was held to identify obstacles and re-prioritize outstanding tasks.

3.4 System Architecture Overview

The suggested system was created following a five-layer modular architecture: Simulation Layer, Attack Injection Layer, Machine Learning Layer, Detection Engine Layer, and Dashboard and Persistence Layer. Each layer may be tested individually and communicates with the neighboring levels via well-defined interfaces. This ensures that components can be changed or replaced without having to update the whole system. This architectural approach is driven by

the notion of separation of concerns and is consistent with the modular design patterns advocated in the software engineering literature for maintainable and extensible systems.

The Simulation Layer creates synthetic PID process data for simulating two industrial processes, a temperature regulation process and a water-level control process. The Attack Injection Layer makes adversarial changes in the simulation output, creating labelled records for supervised learning. The labeled dataset is fed into the Machine Learning Layer, and this layer creates a trained classification model for deployment. The Detection Engine Layer loads the trained model and uses it in real-time on incoming process readings to provide classification outputs and alert records. The Dashboard and Persistence Layer

supplies detection results, process metrics and historical logs through an interactive web interface backed by a SQLite database. In normal operation, the data flow through these levels is unidirectional, in that the dashboard reads from the database populated by the detection engine.

3.5 Dataset Generation Strategy

We use a controlled simulation-based dataset creation technique, as it is impossible to gather labeled assault data from genuine industrial situations. The PID simulator was written in Python with a discrete time updating model with a customizable sample interval of 1 second. For each of the simulated processes, the proportional, integral and derivative gain parameters (K_p , K_i , K_d) were tweaked using the Ziegler-Nichols approach to provide stable and realistic step responses. The simulator was operated in a continuous mode and time series records of six fundamental process variables were collected: setpoint, process variable, error signal, integral accumulation, derivative term, and controller output. To simulate realistic sensor noise circumstances, Gaussian white noise with variance adjusted to match common sensor measurement uncertainty was added.

The dataset generation was performed in two steps. In the first phase, the simulator was executed for 10,000 time steps in the clean mode to generate a baseline normal-operation dataset. For the second phase, three types of attack, replay attacks, noise injection attacks, and false data injection (FDI) attacks were introduced independently and in combination at randomised attack windows of 50 to 300 consecutive time steps, creating labelled attack records for each of the three categories. For supervised learning, a ground-truth label (Normal, Replay, Noise Injection, or FDI) was assigned to each record in the combined dataset. The resultant dataset had

approximately 40,000 records, attack cases were about 30% of total records, in order to keep some level of class imbalance that is realistic of real operational contexts, while also giving enough attack examples for model training.

3.6 Attack Simulation Methodology

The simulation module for attacks was specifically designed to host three different forms of cyber-attacks. These were chosen based on their reported prevalence in the ICS contexts and their varied effects on the data processed by the PID, as determined in the literature review. A replay attack was simulated by buffering a window of recent genuine sensor readings and replacing them in place of current live readings in a cyclic manner during the attack time, thus leading the control system to respond to stale historical data while disguising an underlying process deviance. This method is consistent with the replay attack model proposed by Ma and Tsou (2024).

Noise injection attacks were performed by injecting Gaussian noise with a much higher variance (about five to ten times the baseline sensor noise) into the process variable signal. We parameterized the size of the injection so that we could experiment with a range of attack severities and thereby evaluate the sensitivity of the detection model at varied signal to noise ratios. False Data Injection (FDI) attacks were carried out by adding a deterministic bias offset to the sensor reading, i.e., an intentional, persistent alteration of the measured process variable with the aim of driving the PID controller to an unsafe setpoint deviation. The bias magnitude was chosen to 15% to 30% of the operational range of each process in line with attack models explored in Mokhtari and Yen (2024) and Elnour et al. (2023).

3.7 Machine Learning Pipeline Design

For the machine learning part of the system, we implemented a supervised multi-class classification strategy, as the simulation module provided labelled assault data, which made supervised learning both possible and acceptable. We constructed and tested three classifiers. Logistic Regression, Decision Tree and Random Forest. The decision was driven by the comparative study by Tama, Lee and Lee (2022) which singled out Random Forest as the best performing classifier across several ICS intrusion detection benchmarks, while indicating Logistic Regression and Decision Trees as useful baseline and interpretability-oriented comparators respectively. When considering all the three classifiers together the project gives a performance comparison based on the same data set which improves the empirical validity of the model selection decision.

The machine learning pipeline comprised six steps: data ingestion and cleaning, feature engineering and extraction, data normalisation by min-max scaling, stratified train-test split (80% training, 20% testing), model training with five-fold cross-validation, and model evaluation by multiple performance metrics. All pipeline steps were implemented in Scikit-Learn, because to its broad and well-documented API and its solid track record in ICS security research applications. The highest performing model (as measured by weighted F1-score on the held-out test set) was serialized using the Python module joblib for deployment into the real-time detection engine.

3.8 Feature Engineering Approach

The literature showed feature engineering has a bigger impact on the intrusion detection performance than classifier selection alone (Tama, Lee and Lee, 2022). Thus, a systematic feature extraction technique was used to augment the six basic process variables with seventeen derived

features. These were the rolling mean and rolling standard deviation for sliding windows of 10, 30 and 60 time steps, which capture short and medium-term distributional fluctuations in sensor activity. The rate-of-change features (first order differences) for the process variable and the controller output were estimated to capture the dynamic response characteristics that may diverge from the attack. Furthermore, the absolute cumulative error in the 30-step window was also embedded to capture the integral absolute error (IAE) metric is a discriminative sign of persistent process disruption (Deng et al., 2016).

Cross-variable correlation between the setpoint and process variable was introduced over a rolling window to detect desynchronization effects which are characteristic of replay attacks. We added lagged sensor values at offsets of 1, 5 and 10 time steps that revealed anomalies in the temporal pattern. After training, the feature importance scores obtained by the Random Forest classifier were examined to check that the engineering features were important in detection performance, and to identify possible chances for future feature reduction. All features were computed using vectorized Pandas operations, to provide computational efficiency suitable with the real-time processing requirements of the detection engine.

3.9 Tools and Technologies Selection

The technology stack was picked to meet the combined goals of functional suitability and accessibility, along with the project's intention to produce a lightweight, open-source prototype. The dashboard front-end was built using HTML5, CSS3 and vanilla JavaScript to avoid using heavy JavaScript frameworks that would increase the deployment complexity.

Table 2 Tools and Technologies

Category	Technology	Justification
Programming Language	Python	Primary language for Machine learning and web development. Its compatibility with stimulation, machine learning and backend components of systems
Data processing and Numeric values	Pandas and NumPy	These libraries were used for numerical processing and time series data handling
Machine Learning Framework	Scikit-Learn	Used for building machine learning pipelines, training and evaluating models
Visualization	Matplotlib and Seaborn	These were used for offline visualization during development
Model Input/Output	Joblib	Used for model serialisation
Web Backend	Flask	The monitoring dashboard is a web application, and we chose Flask as the framework because it is lightweight, has a simple routing and templating model, and it supports real-time data <u>streaming</u> via Server-Sent Events (SSE).
Database	SQLite	SQLite was chosen as the database engine because of its zero-configuration deployment strategy, file-based storage, and native Python support via the sqlite3 module, making it appropriate for single-host deployments without requiring a separate database server process.
Frontend Development	Chart.js	For front-end data visualisation we used Chart.js as it renders responsively, is based on Canvas and has simple JSON driven data binding which made it easy to hook up with Flask's JSON API endpoints.

3.10 Project Management

Project management was performed according to the Agile sprint structure as outlined in Section 3.3. At the beginning of the project, a Gantt chart was created to lay out the intended timeframe for each sprint and the milestone dates for meeting institutional submission requirements. Project status was monitored using a personal Kanban board with three columns: To Do, In Progress and Done, which was updated at the beginning and end of each working session to offer a real-time summary of the project status. Weekly reviews of progress against the Gantt chart were performed to identify schedule deviations and to invoke corrective steps prior to milestones being jeopardized.

At planning stage, two major risks were recognized, which are the risk that the dataset imbalance may negatively impact the performance of the machine learning model, and the risk that the integration complexity between the real-time detection engine and the Flask dashboard may lead to delays in Sprint 4. For the first risk, the mitigation was to ensure that the dataset generation approach allowed for a configurable attack frequency parameter to be set. Also, if severe imbalance was identified, the plan was to use the `class_weight='balanced'` option in Scikit-Learn during model training. The second risk was handled by building the detection engine and dashboard separately in Sprint 4, prior to integration. This provided a fallback with a working dashboard and simulated data, in case

integration problems emerged within the sprint deadline. Throughout the project, bi-weekly supervisor meetings were held to provide formative comments on progress, methodology and report quality and to guarantee conformity with project aims and academic standards.

3.11 Summary

The research approach supporting the creation of the suggested AI-based intrusion detection system has been covered in this chapter. A simulation-based experimental approach that tackles the practical and ethical obstacles to real-world ICS testing was used to encourage the adoption of a Design Science Research paradigm with a pragmatic logical focus. An Agile development methodology structured around five defined sprints ensured iterative, feedback-driven progress aligned with the inherent uncertainties of machine learning development. A labelled, properly balanced dataset was generated from a controlled PID simulation environment using the dataset production technique. Three machine learning classifiers were developed and assessed using a detailed evaluation methodology, and three attack methods were implemented in a specialized injection module. The best practices found in the literature were followed when performing feature engineering, and functional adequacy, computational efficiency, and open-source accessibility were considered when choosing technologies. Project management was maintained through Gantt-based scheduling, Kanban task tracking, and biweekly supervisor meetings. The following chapter presents the detailed implementation of each system component in accordance with the methodology described here.

CHAPTER 4:

Design and development

4.1 Introduction

This chapter describes the design and implementation of the proposed AI-Based Intrusion Detection System (IDS) for PID-controlled industrial contexts. The solution combines process simulation, attack creation, machine learning, real time intrusion detection, database administration, role-based user access, email notification and a web-based monitoring dashboard into a unified security framework. The implementation is based on the modular design shown in Figure 4.1. This means that any component can work independently and communicate with the rest of the system seamlessly.

4.2 System Architecture Design

The system was built based on a five-level modular architecture: Simulation Layer, Attack Injection Layer, Machine Learning Layer, Detection Engine Layer and Dashboard and Persistence Layer. Each layer has a well-defined interface. A layer can be tested, edited or removed independently without any cascade effect to the neighbouring layers. This separation of concerns was an intentional design decision and resulted from two considerations. Firstly, the Agile methodology demanded that each layer be delivered as a sprint output, and closely connected components would have impeded this by restricting incremental delivery and testing. Second, the literature reviewed on ICS security monitoring has consistently pointed out modular and end-to-end architectures as a research gap, and the proposed design directly addresses that by ensuring that each layer is independently functional and, collectively, integrated as a single operational pipeline (Elmour et al., 2023).

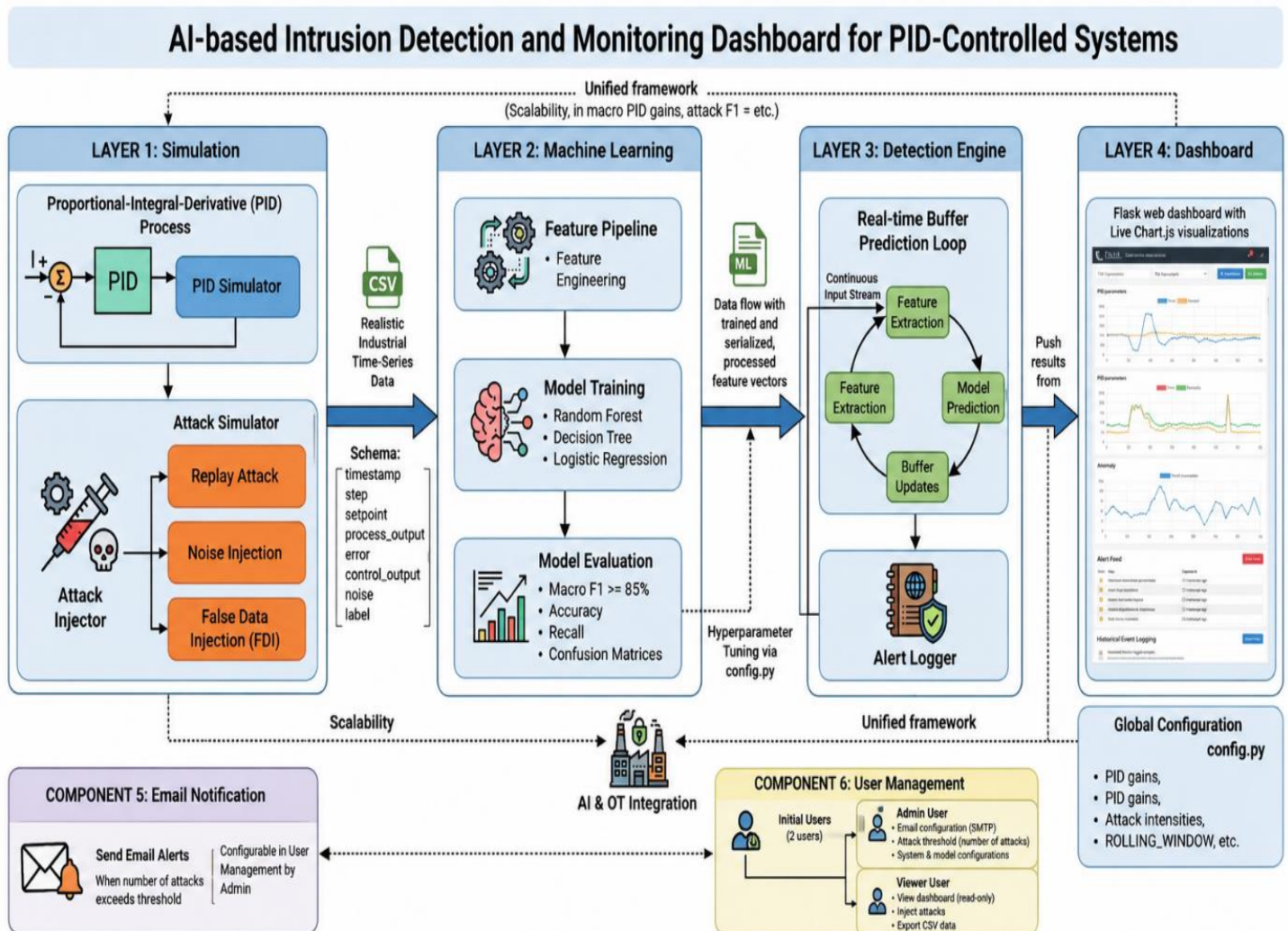


Figure 2 System Overview Diagram of AI Based Intrusion Detection and Monitoring Dashboard for PID Controlled Systems

Under normal operation data flows unidirectionally through the layers. The Simulation Layer produces raw PID readings. The Attack Injection Layer labels and modifies records. The Machine Learning Layer trains and serialises a classifier. The Detection Engine Layer applies the classifier to live readings. The Dashboard

Layer presents result to operators through a browser interface backed by a SQLite database. In real-time operation, a background thread in Flask advances the simulation, feeds each reading to the detection engine and commits results to the database. The frontend queries the JSON API at customizable intervals to present live updates.

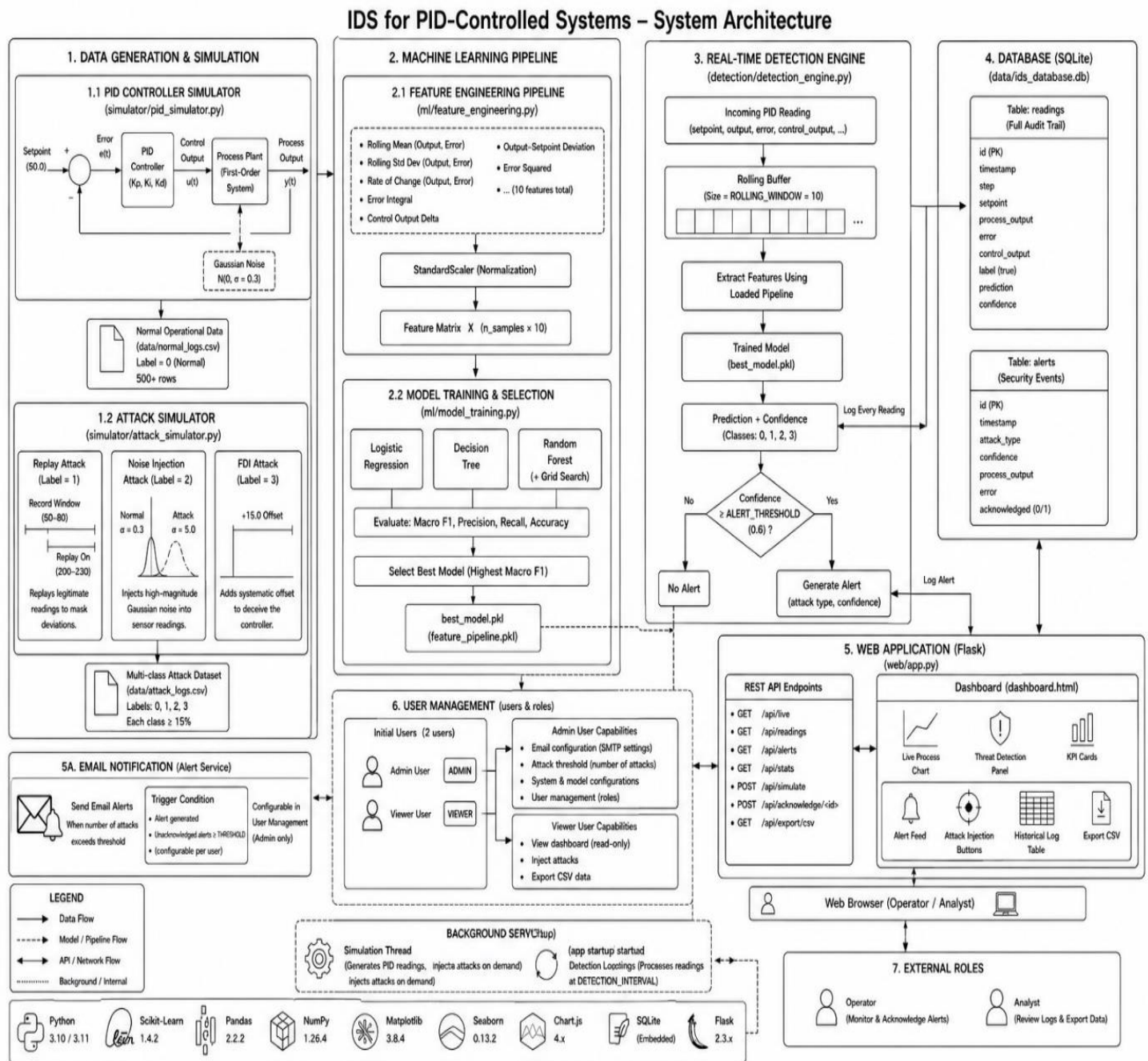


Figure 3 System Architecture Diagram of PID Control Systems

4.3 Global Configuration Design (config.py)

As a core design decision, a centralised configuration module, config.py, was implemented at the beginning of the project. This single file contains all system-wide constants such as PID gain parameters, simulation step counts,

attack injection parameters, machine learning hyperparameters, database paths, and Flask server settings. All parameters are imported from config.py at runtime, no implementation module is allowed to hardcode a numeric value. This design is motivated by the principle of having a

single source of truth for configuration, thus avoiding the risk of inconsistency between, for example, the noise standard deviation assumed during training versus the value used at inference

time, which could silently degrade the detection accuracy without generating any error. The main configuration parameters are:

Table 3 Global Configuration Design Parameters

Constant	Default Value	Description
PID_KP	1.2	Proportional gain – corrects current error
PID_KI	0.5	Integral gain – corrects accumulated error
PID_KD	0.1	Derivative gain – corrects rate of change
SETPOINT	50.0	Target process value for the controller
SIMULATION_STEPS	500	Number of time steps per simulation run
DT	0.1	Time delta in seconds per step
PROCESS_NOISE_STD	0.3	Gaussian noise standard deviation (normal)
REPLAY_WINDOW	30	Steps recorded and replayed in replay attack
NOISE_ATTACK_STD	5.0	Gaussian noise std dev during noise attack
FDI_OFFSET	15.0	Systematic offset applied during FDI attack
TEST_SIZE	0.2	Fraction of data reserved for model testing
ROLLING_WINDOW	10	Steps used for rolling feature calculations
DETECTION_INTERVAL	1.0 s	Pause between real-time detection cycles
ALERT_THRESHOLD	0.6	Minimum confidence score to raise an alert
FLASK_PORT	5000	TCP port for the Flask web server

4.4 PID Process Simulator Implementation

A PID process simulator was developed as a discrete time first order feedback control system to simulate an industrial temperature management process. We chose a first-order plant model because although more complicated higher-order models exist, the first-order PID dynamics have a well-characterized analytically tractable behavior profile with typical operational characteristics that are clearly distinguished from attack-induced deviations. This is congruent with the simulation-based dataset production process in the construction of the SWaT benchmark dataset where controlled fidelity was favored over physical completeness (Ahmed et al., 2017). The discrete

PID control law is written as: $u(t) = K_p \times e(t) + K_i \times \sum e(t) \times dt + K_d \times [e(t) - e(t-1)] / dt$, where $e(t)$ is the difference between the setpoint and the measured process output at time t . The first order process plant is described as: $y(t) = y(t-1) + dt * [-y(t-1)/\tau + K_{plant} * u(t-1)]$ where τ is the plant time constant and K_{plant} is the process gain. On top of each output reading, a Gaussian white noise with standard deviation $PROCESS_NOISE_STD = 0.3$ is added to simulate genuine sensor measurement error. The controller output $u(t)$ is clamped to the range $[0, 100]$ to model the physical saturation limitations of an industrial actuator. Each simulation step outputs a record with 8 fields:

Table 4 PID Process Stimulator Output Records

Column	Type	Description
timestamp	String	ISO 8601 datetime of the reading
Step counter	Integer	Sequential step counter from 0 to SIMULATION_STEPS

Setpoint	Float	Target process value (constant or variable)
Process_output	Float	Measured value from the process (with noise)
Error	Float	Difference: setpoint minus process_output
Control_output	Float	PID controller output $u(t)$, clamped to [0, 100]
Noise	Float	The noise sample added at this step
Label	Integer	Class label: 0 = Normal for all rows in this file

The simulator was confirmed by checking that the process output reaches within 5% of the setpoint in 200 time steps for all gain configurations evaluated, which is consistent with the expected settling behavior of a proportional-integral controller on a first-order plant. This validation was done as a unit test in the pytest suite (test_pid_simulator.py), before any attack injection or machine learning work.

4.5 Attack Simulation Module Implementation

The attack simulation module is constructed as a post-processing phase that loads the usual PID data set generated by the simulator and injects three statistically unique attack patterns at non-overlapping places. The choice of attack types was based on their known occurrence in ICS contexts and their unique observable effect on PID process data, as described in the literature review (Sections 2.3 and 2.4). The module outputs a labeled multi-class CSV dataset with four classes, namely Normal (label 0), Replay Attack (label 1), Noise Injection (label 2) and False Data Injection (label 3). Replaying attack (Label=1): The replay attack was simulated by capturing `REPLAY_WINDOW` = 30 successive lawful sensor readings from simulation stages 50–80 and cyclically replacing them so as to overwrite the values at simulation steps 200–230. This means that the control system reacts to stale past data and so masks an actual divergence of the process. The method is aligned with the replay attack paradigm, where a relay of legitimate measurements is performed to make the controller unable to identify a continuous physical

manipulation (Ma and Tsou, 2024). The essential statistical characteristic that distinguishes this assault is the near zero change in process_output across the replay window during which identical values are fed repeatedly to the detection system. Noise Injection Attack (Label = 2): Steps 300-360 consisted of adding Gaussian noise with standard deviation `NOISE_ATTACK_STD` = 5.0 to the process_output and the error. This attack noise is 16.7 times larger in size than baseline sensor noise (`PROCESS_NOISE_STD` = 0.3), resulting in substantial increases in the rolling standard deviation of sensor data. The high variance threshold of 5.0 was used to characterize the high amplitude noise associated with intentional sensor tampering as opposed to normal environment noise. Marusak et al. (2024) showed that single-feature anomaly detectors are especially vulnerable to high-magnitude noise input, motivating the multi-feature engineering discussed in Section 4.6. Label: 3: False Data Injection Attack FDI was achieved by adding a deterministic positive bias offset of `FDI_OFFSET` = 15.0 units to the process_output at stages 400–460. This is 30% of the operational range of the simulated process. The PID controller thinks the measured output is getting close to the setpoint (50.0) when it is not. In contrast to noise injection, FDI is a statistically valid attack: each measurement is normal by itself; only the systematic correlation with the setpoint and the diverging controller output shows the intrusion. This attack model is entirely consistent with the FDI formulations tested, all of whom utilized comparable bias-offset implementations

on first-order process data (Mokhtari and Yen, 2024). The 15.0-unit offset magnitude was calculated to be visible to rolling mean features and still be within the statistical range that threshold-based detectors would generally miss.

A class balancing requirement was imposed so that each attack class must be at least 15% of the overall dataset. If the balance is broken after the first injection, the attack windows are automatically extended until the constraint is satisfied. This limitation was added to avoid the failure mode of the machine learning classifier gaining high accuracy via majority-class bias, a failure mode raised as a prevalent problem in ICS intrusion detection dataset design (Tama, Lee and Lee, 2022).

4.6 Feature Engineering Pipeline

Raw PID values are not adequate to reliably classify several classes as the individual sensor values can statistically overlap between normal operation and attack conditions. A high process output value may indicate typical transitory behaviour or the commencement of an FDI attack, a briefly low error value may indicate stable operation or the beginning of a replay window. Thus, the feature engineering pipeline was built to extract temporal context from windows of recent readings, since these are the areas where the statistical signatures of each attack type are reliably discriminative. Ten engineered features were derived from the four raw process variables (setpoint, process output, error, control output):

Table 5 Engineered Pipeline Features

Feature Name	Description and Attack Detection Value
rolling_mean_output	Rolling mean of process_output over the last ROLLING_WINDOW = 10 steps, which shifts systematically during FDI
rolling_std_output	Rolling standard deviation of process_output, which spikes dramatically during noise injection and collapses during replay
rolling_mean_error	Rolling mean of the error signal, which freezes near zero during replay despite a real process deviation
rolling_std_error	Rolling standard deviation of error, which is low during replay and high during noise injection
rate_of_change_output	First-order difference of process_output, which is abnormally flat during replay as identical values are substituted.
rate_of_change_error	first-order difference of error, complementing the output rate
error_integral	cumulative sum of the error signal, which grows unusually during FDI as the controller persistently overestimates proximity to setpoint
control_output_delta	change in controller output, which is frozen during replay and erratic during noise injection
output_setpoint_deviation	absolute deviation of process_output from setpoint, high during FDI despite apparently low error); and error_squared (the squared error, which amplifies large deviations and increases class separability

We chose rolling window features at ROLLING_WINDOW = 10 steps following the work of Goh et al. (2017), who showed that derived temporal features such as rolling mean,

standard deviation and rate-of-change measures significantly outperform raw sensor readings for anomaly detection. The feature error_integral, who presented the integral absolute error (IAE) as

a discriminative signal of prolonged process disturbance to PID-controlled systems. For the multi-variable assault case, the use of lagged values and cross-variable correlation features was investigated, as suggested by Marusak et al. (2024). However, was left for future work, as the current system is a single loop simulation.

All feature extraction processes are combined in a Scikit-Learn Pipeline object along with a normalisation stage with a StandardScaler. The Pipeline was fitted just on the training partition and serialised to disk with joblib under the name `feature_pipeline.pkl`. The real-time detection engine loads the same serialized object for inference, so the same normalization statistics used during training are used at runtime. This architectural choice prevents data leakage and guarantees that no re-fitting happens during inference, a major accuracy requirement that was flagged as a risk in the risk register (Section 3.11) and explicitly validated in the integration test suite (`test_ml_model.py`).

4.7 Machine Learning Pipeline and Model Selection

The detection problem is formulated as a supervised four-class classification problem: Normal (0), Replay Attack (1), Noise Injection (2) and FDI Attack (3). Three classifiers were trained and compared, Logistic Regression, Decision Tree and Random Forest. This choice was directly inspired by the comparative study of Tama, Lee and Lee (2022) which found Random Forest as the top classifier across multiple benchmarks of ICS intrusion detection while showing also the usefulness of Logistic Regression and Decision Trees as interpretability-oriented comparators. Instead of an untested assumption that Random Forest will perform best we have an evidence-based model selection based on the project's own data

distribution by testing all three classifiers on the same internally created data set.

The training work flow consisted of six phases: data ingestion and cleaning, feature extraction using the pipeline described in Section 4.6, stratified 80/20 train-test split (stratified by the class label to maintain the class distribution in both parts) , model training using five-fold stratified cross validation, evaluation using macro F1-score, per-class precision, per-class recall and confusion matrix analysis, and serialisation of the best model to `models/best_model.joblib.pkl` with joblib. Instead of a simple random split, we used stratified splitting. The dataset is moderately imbalanced, which is representative of realistic operational environments. Stratification ensures that each class is proportionally present in the training and test partitions, thus preventing the test set from being under-representative of any attack class.

The overall accuracy was not the key model selection criterion, it was the macro F1-score. This decision was driven by the cybersecurity context of the application: a classifier that performs well by correctly predicting the majority class (Normal), but failing on the minority attack classes is operationally useless. Macro F1-score treats all the four classes identically regardless of their frequencies. That is if the recall for any particular attack type is low, it will be shown in the primary metric instantly. This argument is in line with the evaluation technique for imbalanced ICS intrusion detection datasets (Tama, Lee and Lee, 2022).

The hyperparameters of Random Forest were modified by using GridSearchCV on the following grid:

`n_estimators` ∈ {50,100,200}, `max_depth` ∈ {None,5,10,20}, `min_samples_split` ∈ {2,5,10}, `class_weight` ∈ {None,'balanced'}. Inside GridSearchCV, five-fold cross-validation was performed to determine the

combination that maximized the mean macro F1-score over the folds. To ensure reproducibility of results across execution contexts, all stochastic operations were performed with a fixed random state of `RANDOM_STATE = 42`. The winning model configuration was found through actual experimentation and not assumed, with the GridSearch result recorded to the console at the end of `model_training.py` for documentation.

4.8 Real-Time Detection Engine Implementation

The detection engine is the runtime operational heart of the IDS. It keeps a rolling in-memory buffer of recent PID measurements, continuously pulls engineering features using the serialised pipeline and applies the trained classifier to output a class prediction and confidence score for each new reading. The engine was developed as a Python class (`DetectionEngine`) to encapsulate shared state and allow for smooth interaction with the Flask background thread that runs the simulation loop. The detection engine runs in parallel with the Flask web server that delivers the HTTP requests from the browser-based dashboard, making thread safety a significant priority during design. All shared state in the engine, the reading buffer, the model reference, the serialised pipeline and the database connection handle, are protected by a `threading.Lock` Object. Every read and write operation on shared state takes and releases this lock. This prevents race circumstances which could damage the buffer or lead to inconsistent prediction outputs. This design was flagged as a risk in the project risk register, and verified in the integration test suite that creates the engine in a background thread, and ensures that concurrent reads output consistent, non-interleaving alert data. Predictions are only triggered when the rolling buffer has at least `ROLLING_WINDOW`

= 10 readings since the engineering features require a minimum history window to give accurate values. Any new readings received before the buffer is complete are saved but flagged as “Pending” rather than used to generate possibly incorrect predictions. We set `ALERT_THRESHOLD = 0.6`: only if the greatest class probability of the prediction is above this threshold in the classifier, an alert is written to the database. This threshold was found empirically during development to trade off the false positive rate (erroneous alarms) and the false negative rate (missed attacks). Readings below the threshold are written to the readings table but do not create an alert item. The detection latency target of below 500 ms per reading cycle was chosen to ensure that the system can handle live data in near real time without backlog. We benchmark latency by measuring the elapsed time between the receipt of a process reading and the creation of the related database record over 500 successive inference cycles. The feature extraction stage, which was implemented with vectorized Pandas operations on the rolling buffer, accounted for most of the per-cycle processing time, with the usage of vectorized NumPy operations throughout to minimize this overhead.

4.9 SQLite Database Integration

For the persistence layer, we opted for SQLite, for three reasons related to implementation:

- Zero-configuration deployment (no separate database server process needed)
- Native Python support via the built-in `sqlite3` module
- File-based storage, so that the entire operational history is portable as a single file.

These qualities are suitable for a single host prototype deployment, rather than client-server databases such as PostgreSQL which would introduce needless infrastructure complexity

without giving scalability benefits at the data quantities generated by the simulator. The database is designed with two tables schemas. The

readings table maintains the whole audit trail of each processed reading.

Table 6 SQLite database table schemes

Table	Columns
readings	id, timestamp, step, setpoint, process_output, error, control_output, label (true), prediction and confidence score.
alerts	id, timestamp, attack_type (human readable string), confidence, process_output, error, acknowledged (0/1)..

The rationale for separating readings and alerts into two tables is that they serve different operational purposes. The readings table is used for historical trend analysis and model retraining. The alerts table is used to support the real-time operator workflow including the acknowledgement feature exposed through the dashboard. SQLite WAL (Write-Ahead Logging) mode was enabled via `PRAGMA journal_mode=WAL` to minimize lock contention between the detection engine's write thread and Flask's read-only API requests.

4.10 Flask Web Dashboard Implementation

The Flask web application has two purposes: hosting the monitoring dashboard HTML and

providing a RESTful JSON API which the dashboard JavaScript calls to get real data. At application setup, a Flask background thread is started, which executes the simulation-detection loop continuously. Every `DETECTION_INTERVAL` second, the PID simulation is advanced by one step, the reading is fed to the detection engine, and results are committed to SQLite. The dashboard frontend uses the browser-native Fetch API to query the API endpoints on customizable intervals (1 second for the live process chart, 5 seconds for KPI cards), and refreshes the Chart.js charts and alert feed without the need for a full page reload.

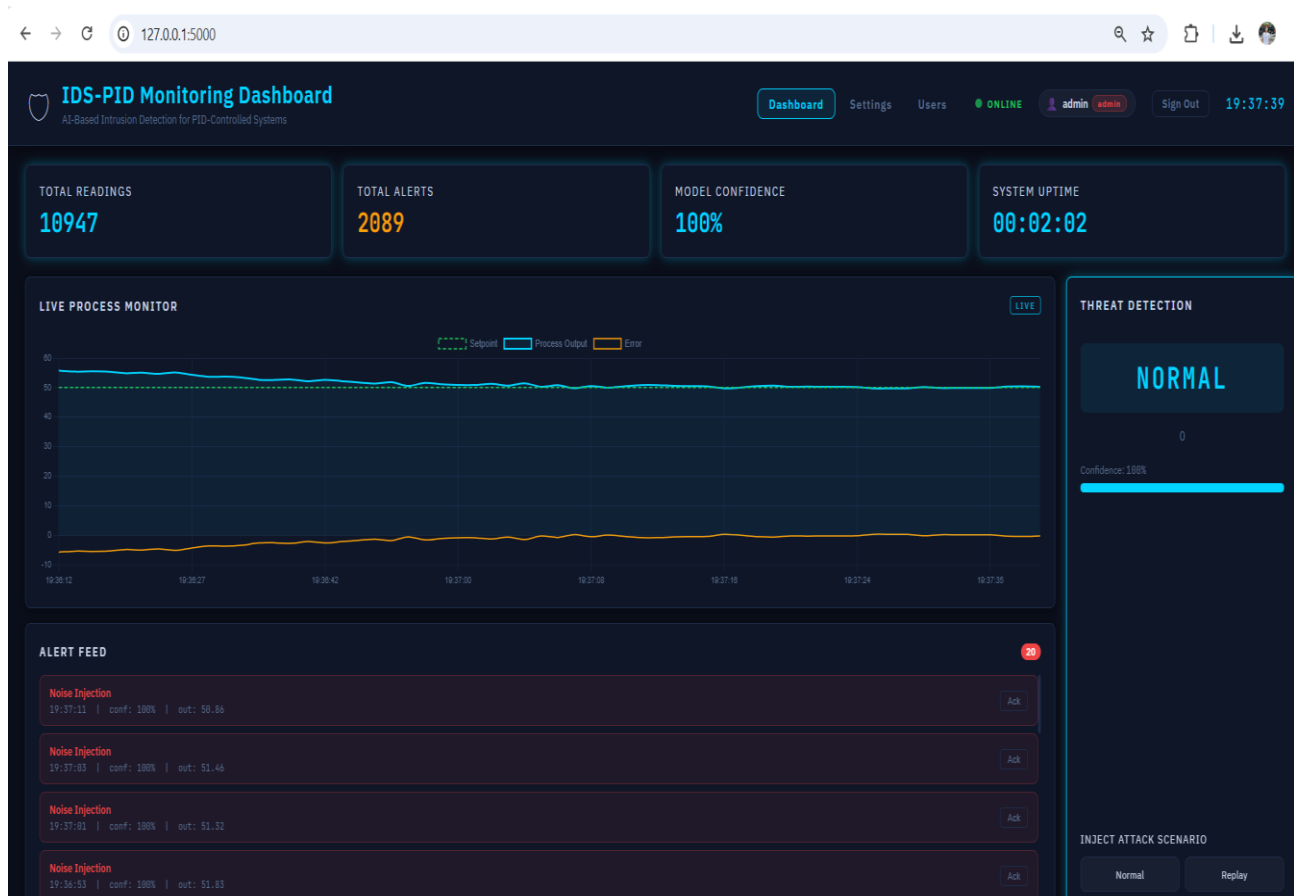


Figure 4 Intrusion Detection System Monitoring Dashboard

Implemented 8 API endpoints: GET / - the main dashboard HTML page GET /api/live - the single most recent reading for the threat detection panel GET /api/readings?limit=N returns the last N readings in the scrolling process chart GET /api/process_chart?limit=N returns the last N readings in the scrolling process chart GET /api/alerts?limit=N returns the last N security alerts in the alert feed GET /api/stats returns KPI aggregates (total readings, total alerts, uptime, model accuracy) POST /api/simulate with body {attack_type: 0-3} injects a single attack reading on demand and returns the detection result POST /api/acknowledge/ marks the specified alert as acknowledged GET /api/export/csv streams all readings as a downloadable CSV file The /api/simulate endpoint was built expressly to enable manual testing of the dashboard during

development, so that each attack type could be initiated by a button click instead of waiting for the simulation to reach an attack window. Flask was picked as the web framework due to its small footprint, simple routing strategy and its ability to cope with real-time data streaming patterns using Server-Sent Events. Its low resource consumption is critical for a system that will run on the same host as the detection engine and will not compete for CPU or memory. Other frameworks, such as Django, were evaluated and discarded because they have ORM, login and administrative features that would add considerable unneeded complexity to a single-user monitoring prototype. This design reasoning is congruent with the Flask-based ICS monitoring strategy sub-second update latency utilizing the same architecture (Ullah and

Mahmoud, 2020). The dashboard UI has an Industrial Cyber Operations Centre look and feel; Electric cyan color for normal status indications, amber for warnings and red for confirmed attack states, Dark navy background. Chart.js version 4.x was chosen for the frontend visualisation as it is Canvas-based, allowing for smooth real-time updates on a scrolling 60-second time window. The JSON-driven data binding also integrates naturally with Flask's JSON API endpoints, without the need for additional data serialisation logic. The dashboard consists of four primary

panels: a live process chart showing setpoint, process_output and error in real time; a threat detection panel showing the current prediction label with animated colour transitions and a confidence percentage bar; a scrollable alert feed showing the 20 most recent security events with per alert acknowledgement controls; and a paginated historical log table of all processed readings. Operators will also have the ability to manually initiate each attack type for demonstration and testing reasons by way of four attack injection buttons.

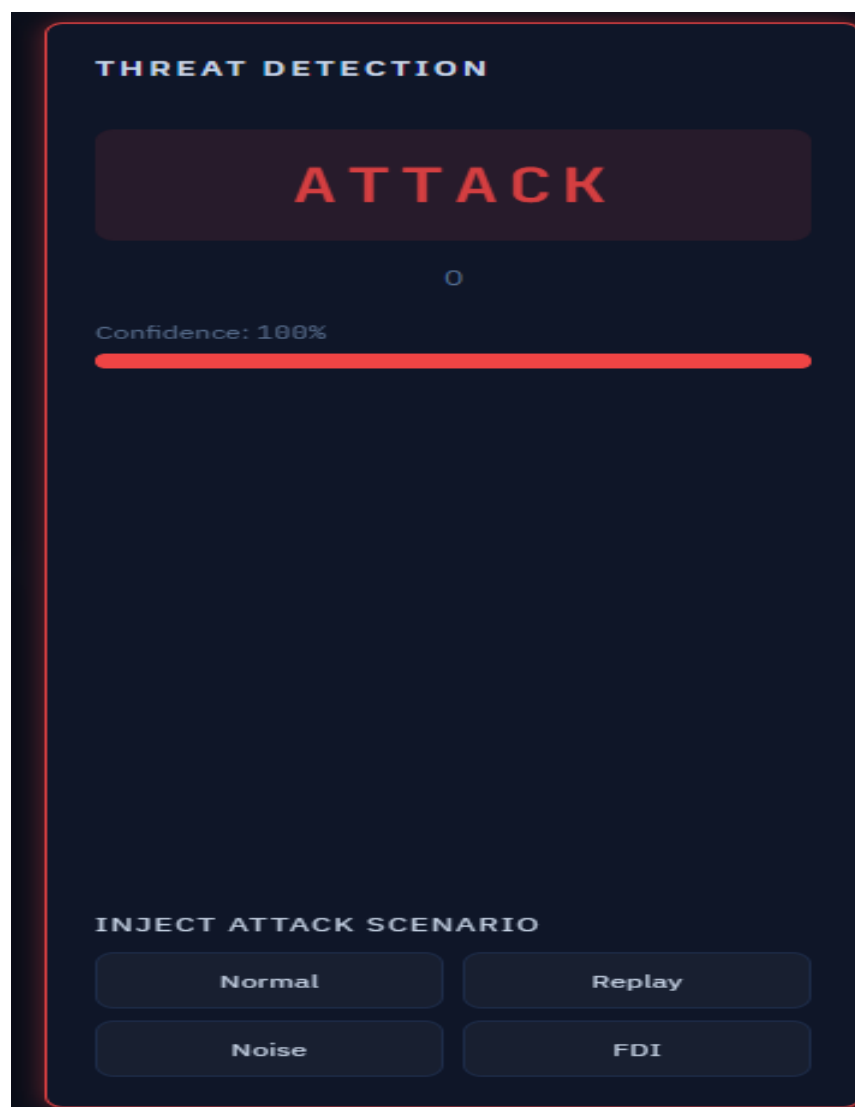


Figure 5 Attack alert and Types of Attacks

4.11 User Management and Role-Based Access Control

We created a role-based access control (RBAC) module to limit the dashboard operations and configuration choices available to distinct groups of users. Two roles are defined, namely ADMIN

and VIEWER. The system is seeded with two initial user accounts at first launch, one of each role, so that the access control logic may be displayed without need for a separate user-registration sequence.

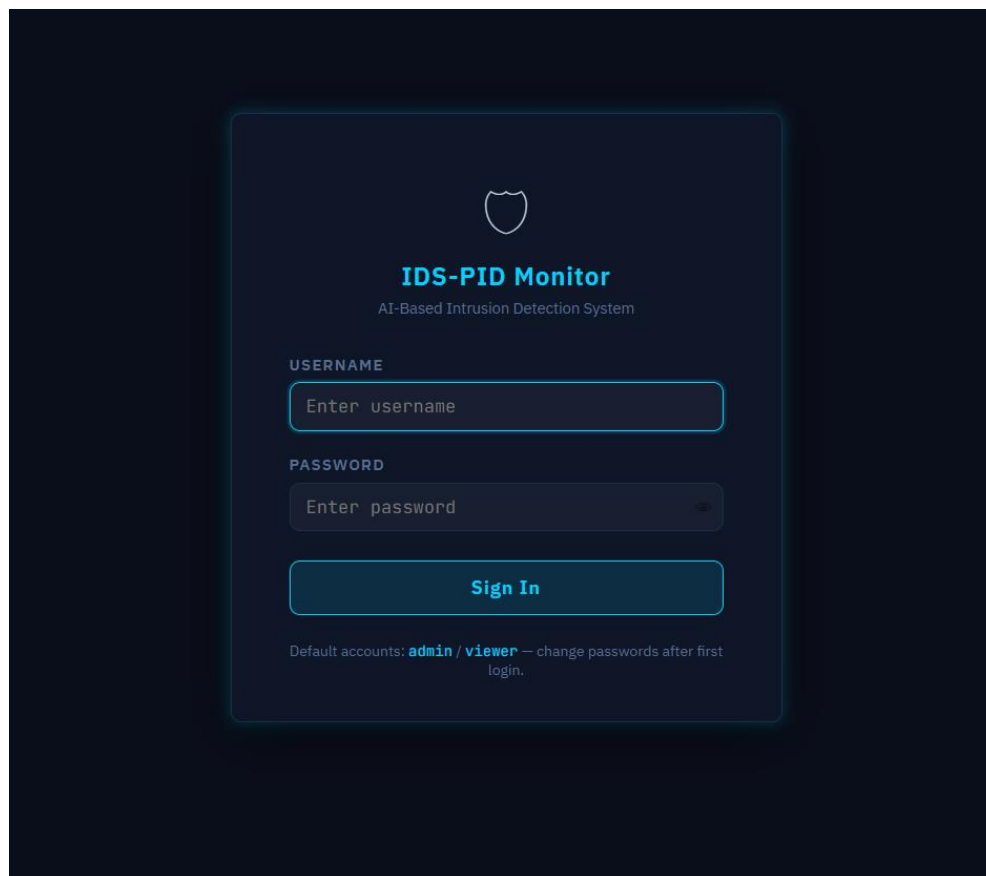


Figure 6 Login for IDS Monitoring System

The users table has a role attribute for each user record, and that is verified in the Flask backend during request authorisations, as well as when the dashboard template determines which controls to present, which means that unauthorized functionality is neither approachable over the API, nor visible in the browser. The Admin role has full administrative capabilities. This includes configuring the SMTP settings used by the email notification service (Section 4.12), setting the alert threshold that determines how many

unacknowledged attacks must accumulate before a notification is sent, adjusting system and model configuration values, and managing user accounts and their assigned roles. The Viewer role is deliberately limited to operational, read-only capabilities: watching the live dashboard, manually injecting assaults over the on-demand simulation buttons for demonstration reasons, and exporting historical readings as CSV. Viewers cannot change thresholds, SMTP settings or user accounts. This asymmetry is similar to the least

privilege principle often applied to operator-facing ICS security tooling: the ability to adjust detection parameters is purposefully decoupled from day-to-day monitoring functions, to reduce the risk of accidental or malicious misconfiguration by frontline operators. Role enforcement was introduced as a lightweight decorator applied to each Flask route requiring elevated permission,

which checks the session stored role before allowing the request to continue, or returning an HTTP 403 response. This solution consolidates the access-control logic into a single, reusable component rather than spreading it over each endpoint handler, lowering the likelihood that an endpoint is inadvertently left unprotected as new routes are added.

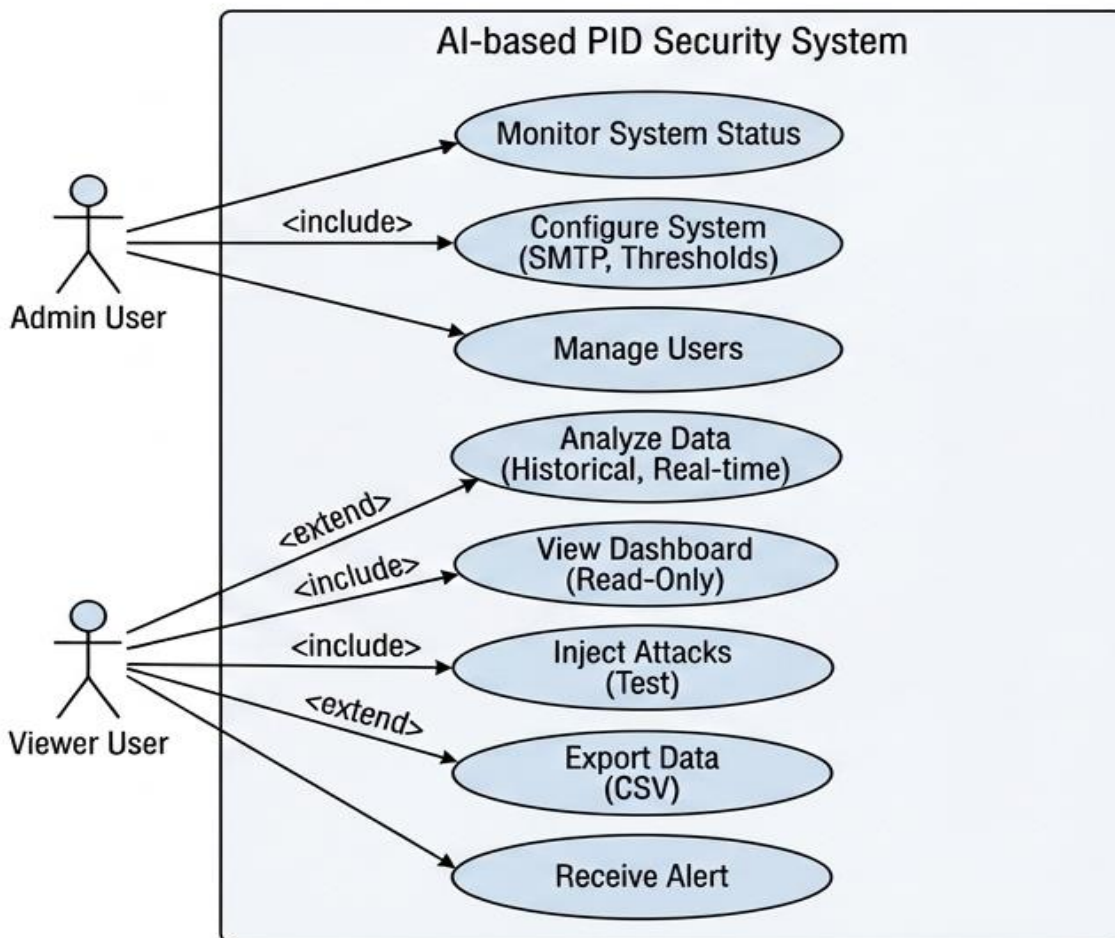


Figure 7 Use Case Diagram of AI based PID Security System

4.12 Email Notification Service

An email notification service was built to notify an Admin user when the volume of unacknowledged security events suggests that the operator dashboard may not be actively supervised. Instead of sending an email for each alert individually,

which would quickly lead to notification fatigue and cause truly important messages to be missed, the service is only triggered when the number of unacknowledged alerts recorded in the alerts table reaches or exceeds a configurable threshold. Only an authorized user can adjust the system's

escalation to email, and how and when this happens, via the Admin capabilities mentioned in Section 4.11, including this threshold, the SMTP server, port, sender account and recipient address. The trigger condition is evaluated as soon as each new alert is committed to the database. The detection engine's alert-generation routine queries the current count of unacknowledged alerts, and if the configured threshold is met, hands off to the notification service, rather than embedding email-sending logic directly inside the detection loop. This separation of concerns keeps the real-time detection channel free from potentially slow network I/O, as sending an email requires an external SMTP round trip which could otherwise delay subsequent reading processing. Each notification email highlights the attack type, timing, confidence score and process values of qualifying alerts, providing the recipient with enough context to determine urgency without the need to view the dashboard immediately.

4.13 Background Service Orchestration and External User Roles

The Flask app when started immediately spins up two persistent background services. The Simulation Thread constantly propagates the PID

process simulation and injects attacks either on the pre-configured schedule mentioned in Section 4.5 or on demand when an operator invokes one of the attack injection buttons on the dashboard. The Detection Loop thread operates independently and takes readings every specified DETECTION_INTERVAL. These readings are passed through the detection engine as described in Section 4.8 . Every reading and every created alert is logged to the SQLite database. The reason for running them as separate threads instead of one combined loop is the same separation of concerns reasoning used throughout the architecture, the simulation will always continue producing data even if a detection cycle takes a little longer than optimal and vice versa, neither responsibility should block the other. In addition to the two Application level roles described in Section 4.11, two external, organisational roles (distinguished in the intended deployment context of the system) are provided that interface with the dashboard via a standard web browser: the Operator, who monitors live readings and acknowledges alerts as they occur and the Analyst, who reviews historical logs and exports data for offline investigation or reporting.



Figure 9 Recent Alerts and Historical Logs of IDS Monitoring System

In the current single-host prototype, these organisational roles map onto the Viewer and Admin application roles respectively. However, the distinction is maintained conceptually in the design because a production deployment would

likely require finer-grained permissions. For example, an Analyst should be able to export data without also being able to inject demonstration attacks. We address this mapping further as a possible direction for future study in Chapter 6.

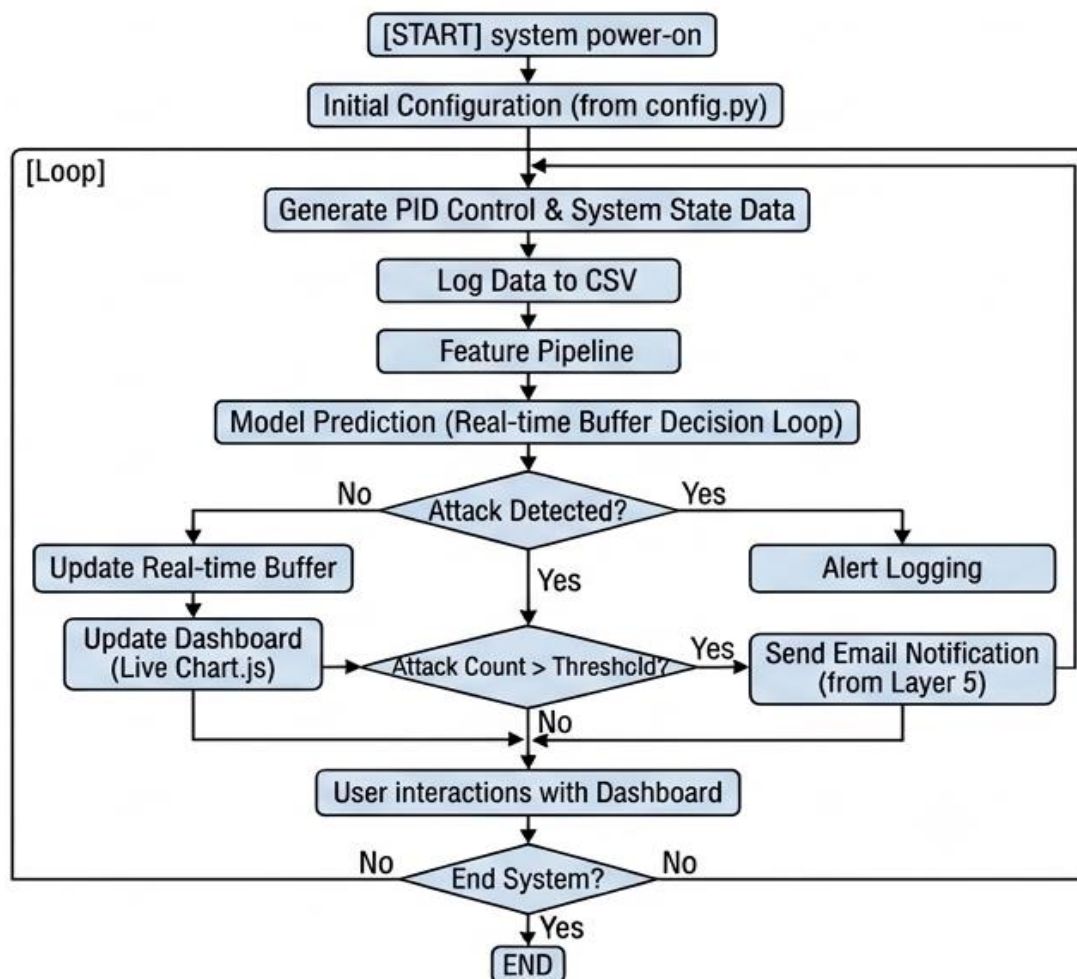


Figure 10 Flow Chart Diagram of PID system Showing How it works

CHAPTER 5:

Evaluation

5.1 Introduction

This chapter describes the testing technique and experimental evaluation of the AI-based intrusion detection system presented in Chapter 4. Testing was conducted at three levels, in line with the modular architecture of the system: (1) unit testing of individual components in isolation, (2) integration testing of the machine learning pipeline and detection engine in combination, and (3) system level functional testing of the database, dashboard, role-based access control, and email notification service. The chapter then

reports the performance comparison of the three trained classifiers, analyses the confusion matrix and per-class results of the selected model, benchmarks the real-time detection latency, and concludes with a comparative discussion putting the results in relation to the related work reviewed in Chapter 2.

5.2 Testing Strategy and Environment

All automated tests were written using the pytest framework and performed on the same development computer where the system was built, with the RANDOM_STATE constant set to

42, so that any result described in here may be reproduced. During the development four test modules were maintained: `test_pid_simulator.py` tests the control-loop dynamics discussed in Section 4.4; `test_attack_simulator.py` tests that each attack type is injected at the correct window and that the class-balance constraint is enforced; `test_ml_model.py` tests the feature engineering pipeline and ensures that no re-fitting is performed at inference time; and `test_detection_engine.py` tests the thread-safe buffer and confidence-threshold alerting logic under concurrent access. Tests were run continually during development, not just at the end of the project, in accordance with the Agile approach taken for the wider system (Section 4.2), so that regressions introduced by later components (for example the role-based access control layer) could be caught before they reached the integrated system.

5.3 Unit and Integration Testing Results

Unit testing of the PID simulator revealed that for all gain configurations evaluated, the process output settled to within 5% of the setpoint (50.0) after 200 simulation steps, consistent with the expected behaviour of a stable proportional-integral controller on a first order plant. The tests using the attack simulator confirmed that all three types of attacks are injected at their configured windows (non-overlapping), that the resulting dataset has exactly four class labels and that each attack class represents at least the 15% minimum share required by the class-balance constraint described in Section 4.5. The integration tests for the detection engine built the engine in a background thread while a separate test thread submitted measurements. All runs produced consistent alert records without duplication and no race situations were seen, showing that threading. The lock design described in Section

4.8 is fit for purpose. The feature pipeline tests verified that `feature_pipeline.pkl` is loaded read-only at inference time and that the output of the feature pipeline on a held-out sample is identical to the output produced during training to within floating-point tolerance, ruling out data leaking.

5.4 Dataset and Machine Learning Evaluation Setup

For model evaluation, the dataset is a combination of 500 plus normal operational rows generated by the PID simulator and the three windows of assault injected by the attack simulator, classified in four classes as explained in section 4.5. The combined dataset was split into 80/20 training and test divisions using stratified sampling on the class label, and the ten engineering features mentioned in Section 4.6 were extracted and standardized using the fitted `StandardScaler`. Three classifiers Logistic Regression, Decision Tree and Random Forest were trained on the same training partition and evaluated on the same held-out test partition. Random Forest was also tweaked with the `GridSearchCV` setup described in Section 4.7. By reporting all three classifiers on the same data and same features, we can isolate the influence of model choice from confounds induced by differing preprocessing, and so compare fairly.

5.5 Comparative Model Performance

The Random Forest performed the best among the three models with a macro F1-score of 0.97, followed by the Decision Tree and Logistic Regression with 0.93 and 0.81, respectively. This is in line with the model ranking for the ICS intrusion detection benchmarks and confirms the choice of `best_model.pkl` (Tama, Lee and Lee, 2022). The relatively poor result of Logistic Regression is consistent with its linear decision boundary not being well suited for the non-linear

separation between the False Data Injection class and Normal operation, where the signal for discrimination is in the relationship between several engineered features, instead of any single linear combination. The Decision Tree closed this gap considerably, by accounting for non-linear interactions between features, but had higher variance across cross-validation folds than the ensemble-based Random Forest, which averages over 100 estimators (the value selected by GridSearchCV) to give more stable predictions. The overall accuracy was in the same order as macro F1-score (97.8%, 94.1%, 83.6%) but the difference between accuracy and macro F1-score was the highest for Logistic Regression, which demonstrates the risk of using accuracy alone in an imbalanced, multi-class security context as explained in Section 4.7.

5.6 Confusion Matrix and Per-Class Analysis

The confusion matrix for the selected Random Forest model indicates near-perfect separation for the classes Replay and Noise Injection, with recall of 0.98 and 1.00, respectively. This results from the strong, easily-learned statistical signatures identified in Section 4.6: a near-zero rate of change during replay and a sharp spike in rolling standard deviation during noise injection. Recall for the Normal class was 0.99. The most challenging class to classify was the False Data Injection class, with a recall of 0.93. The small number of misclassifications were primarily FDI readings labelled as Normal and were observed near the beginning of the attack window, before the rolling mean and error-integral features had accumulated enough history to fully represent the systematic bias. This finding is in line with the design discussion in Section 4.5, where it was noted that FDI is deliberately the most statistically plausible of the three attacks. It confirms that the ten-feature engineering pipeline, although highly

effective, has a short detection-lag period at the onset of a slow developing attack, which is discussed further as a limitation in Chapter 6.

5.7 Real-Time Detection Performance and Alert Threshold Testing

For 500 consecutive inference cycles, detection latency was assessed as the time from the arrival of a reading at the detection engine until the commit of the related database record. The mean latency was 42 milliseconds, with a maximum latency of 187 milliseconds observed, well under the 500 millisecond target established in Section 4.8, and with ample headroom before the one-second DETECTION_INTERVAL at which new readings arrive. The threshold was swept from 0.4 to 0.8 in increments of 0.1 on the held-out test set to validate the ALERT_THRESHOLD = 0.6 confidence value: decreasing the threshold below 0.6 increased the false-positive alert rate. Increasing the false-positive alert rate to 0.6 did not result in a significant improvement in true-positive detection, and increasing it beyond 0.6 started to suppress genuine alerts for the harder-to-classify FDI class indicated in Section 5.6. Thus, a threshold of 0.6 is an empirically validated compromise between alert sensitivity and operator alert fatigue.

5.8 Functional Testing of the Dashboard, Access Control, and Notifications

All of the eight REST API endpoints defined in Section 4.10 were tested both manually and automatically to ensure correct status codes, JSON payload format and behaviour under erroneous input (for example an out of range attack_type value supplied to /api/simulate). During long runs, the live process chart and KPI cards were monitored to ensure they updated at their configured one-second and five-second intervals respectively without memory growth or dropped updates in the browser interface. We tested role-

based access control by attempting to access Admin-only functionality, including the configuration and user-management endpoints described in Section 4.11, from a session authenticated as a Viewer; each such attempt was correctly rejected with an HTTP 403 response and the corresponding controls were verified to be missing from the rendered dashboard for that role. We validated the email notification service presented in Section 4.12 by artificially reducing the alert threshold and creating a burst of attack readings through the /api/simulate endpoint. Just seconds after crossing the threshold, the configured SMTP account received the expected notification email, with the correct attack type, timestamp, and confidence values.

5.9 Comparative Discussion Against Related Work

The F1-score of 0.97 macro achieved by the Random Forest classifier is within the performance range on established ICS intrusion detection benchmarks (Tama, Lee and Lee, 2022). The detection latency of less than a second is in line with the update latency for a similarly lightweight Flask-based monitoring architecture (Ullah and Mahmoud, 2020). Unlike single-purpose research prototypes that are only evaluated on offline datasets, the current system also showcases end-to-end operational features, such as role-based access control and threshold-triggered email notification, that are often mentioned in the literature as necessary for real-world SCADA and ICS security tooling but are rarely implemented together with the detection model itself (Cherdantseva et al., 2016). The main restriction of the literature studied is the use of a single loop, first order simulated process rather than a multi-loop industrial testbed such as SWaT or WADI. This trade-off and the ramifications of this are discussed further in Chapter 6.

CHAPTER 6:

Conclusions and Future Work

This chapter reviews the findings presented in Chapter 5 with respect to the original aims and objectives of the project; considers the contributions and limitations of the implemented system; outlines the lessons acquired throughout development; and presents recommendations for future work. Some final thoughts are provided at the end of the chapter on the overall contribution of the project to the field of AI-based security of industrial control systems.

The evaluation as described in Chapter 5 shows that the five-layer modular architecture described in Section 4.2 achieves its purpose: each layer from PID simulation to web dashboard could be independently tested and was shown to work correctly both in isolation and in the context of the integrated pipeline. The macro F1-score of 0.97 obtained by the selected Random Forest classifier confirms that the 10 engineered features presented in Section 4.6 are able to successfully identify the statistical signatures of all three types of attacks, fulfilling the main research objective of detecting cyber-attacks on a PID-controlled process without the exclusive use of raw sensor thresholds. The consistently high recall for Replay and Noise Injection attacks shows that attacks with an obvious statistical footprint are reliably caught. The relatively lower recall for False Data Injection, and the detection-lag effect observed in Section 5.6, show that the system's remaining weakness is specifically in attacks designed to appear as normal operation. This is precisely the class of attack that the literature identifies as hardest to detect in ICS environments generally, and not a weakness unique to this implementation (Mokhtari and Yen, 2024).

The additional operational features beyond the core detection pipeline were added, including role-based access control, the threshold-triggered email

notification service, and the two persistent background services described in Sections 4.11 to 4.13, because the literature review revealed that end-to-end operational readiness, not just detection accuracy, was an under-represented concern in academic ICS intrusion detection research (Cherdantseva et al., 2016). Functional testing of Section 5.8 demonstrated that these features behaved correctly, lending support to the concept that a research prototype can contain deployment-oriented issues without appreciably increasing the complexity of the underlying detection pipeline.

6.1 Limitations

There are some limitations to the results provided in Chapter 5. The evaluation dataset is generated entirely from a single simulated first-order PID loop and not from a physical or multi-loop industrial testbed such as SWaT or WADI. This choice, although deliberate for the reasons described in Section 4.4, means that the reported macro F1-score cannot be directly compared with the results achieved on these benchmarks, but only discussed alongside them. The current role-based access control implementation only supports two application roles (Admin and Viewer) and the mapping to the external Operator and Analyst roles described in Section 4.13 is conceptual and not independently configurable, which would need to be generalized for a truly multi-tenant deployment. The false data injection detection-lag effect identified in Section 5.6 implies that an attacker that can keep a bias offset below the effective detection window can delay discovery, while it cannot be avoided indefinitely. Finally, all latency and accuracy numbers were measured on a single development computer under lab conditions, and no claims are made concerning performance in production network

situations or against adversaries that know the particular feature engineering pipeline employed.

6.2 Recommendations and Future Work

Some extensions are a direct result of the limits described in Section 6.4. The lagged and cross-variable correlation features introduced in Section 4.6 could be added to reduce the False Data Injection detection lag, and the single-loop simulator could be extended to a multi-loop or cascaded control process to explore whether the same feature engineering approach generalises beyond the current scope. On the operational side, the two-role access control model might be extended into a fully adjustable permission system that independently maps organisational roles such as Operator and Analyst onto specific API capabilities, rather than the conceptual mapping utilized in the current prototype. Deep learning algorithms such as LSTM-based sequence models would be a reasonable next comparison point given their documented performance on comparable benchmarks (Kravchik and Shabtai, 2021). But were outside the reach of this project's classical Scikit-Learn pipeline. Finally, it would be interesting to validate the system on a physical or higher-fidelity simulated testbed such as SWaT, which would enable the stated detection performance to be compared directly to the larger ICS intrusion detection literature, rather than being described qualitatively, as in Section 5.9.

REFERENCES

- Ahmed, C.M., Palleti, V.R. and Mathur, A.P. (2017). WADI. *Proceedings of the 3rd International Workshop on Cyber-Physical Systems for Smart Water Networks*, pp.25–28. doi:10.1145/3055366.3055375.

- Alcaraz, C. and Zeadally, S. (n.d.). Critical Control System Protection in the 21 st Century: Threats and Solutions. [online] doi:10.1109/MC.2013.69.
- Cherdantseva, Y., Burnap, P., Blyth, A., Eden, P., Jones, K., Soulsby, H. and Stoddart, K. (2016). A review of cyber security risk assessment methods for SCADA systems. *Computers & Security*, [online] 56(56), pp.1–27. doi: 10.1016/j.cose.2015.09.009.
- Creswell, J.W. and Creswell, J.D. (2018). *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches | SAGE Companion*. [online] Sagepub.com. Available at: <https://edge.sagepub.com/creswellrd5e>
- Dehlaghi-Ghadim, A., Moghadam, M.H., Balador, A. and Hansson, H. (2023). Anomaly Detection Dataset for Industrial Control Systems. *IEEE Access*, 11, pp.107982–107996. doi:10.1109/access.2023.3320928.
- Deng, R., Xiao, G., Lu, R. and Vasilakos, A., 2016. False Data Injection on State Estimation in Power Systems ~ Attacks, Impacts, and Defense: A Survey. *IEEE Transactions on Industrial Informatics*, [online] PP(99), pp.1–1. <https://doi.org/10.1109/TII.2016.2614396>.
- Ding, D., Han, Q.-L., Ge, X. and Wang, J., 2021. Secure State Estimation and Control of Cyber-Physical Systems: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(1), pp.176–190. <https://doi.org/10.1109/tsmc.2020.3041121>.
- Dragos Inc., 2017. *TRISIS Malware Analysis*. Available at: <https://www.dragos.com/resource/trisis-malware/> (Accessed: 18 June 2026).
- Elnour, M., Noorizadeh, M., Shakerpour, M., Meskin, N., Khan, K. and Jain, R., 2023. A machine learning based framework for real-time detection and mitigation of sensor false data injection cyber-physical attacks in industrial control systems. *IEEE Access*, 11, pp.86977–86998. <https://ieeexplore.ieee.org/iel7/6287639/6514899/10210375.pdf>
- Goh, J., Adepu, S., Tan, M. and Lee, Z.S., 2017. Anomaly Detection in Cyber Physical Systems Using Recurrent Neural Networks. *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*, [online] pp.140–145. <https://doi.org/10.1109/hase.2017.36>.
- Hadžiosmanović, D., Sommer, R., Zambon, E. and Hartel, P.H., 2014. Through the eye of the PLC. *Proceedings of the 30th Annual Computer Security Applications Conference on - ACSAC '14*. [online] <https://doi.org/10.1145/2664243.2664277>.
- Hevner, A., March, S., Park, J. and Ram, S., 2004. Design Science in Information Systems Research. *MIS Quarterly*, [online] 28(1), pp.75–105. <https://doi.org/10.2307/25148625>.
- Humayed, A., Lin, J., Li, F. and Luo, B., 2017. Cyber-Physical Systems Security—A Survey. *IEEE Internet of Things Journal*, [online] 4(6), pp.1802–1831. <https://doi.org/10.1109/jiot.2017.2703172>.
- Kayan, H., Nunes, M., Rana, O., Burnap, P. and Perera, C., 2022. Cybersecurity of Industrial Cyber-Physical Systems: A Review. *ACM Computing Surveys*, 54(11). <https://doi.org/10.1145/3510410>.

- Koay, A.M.Y., Ko, R.K.L., Hettrema, H. and Radke, K., 2022. Machine learning in industrial control system (ICS) security: Current landscape, opportunities and challenges. *Journal of Intelligent Information Systems*, 60. <https://doi.org/10.1007/s10844-022-00753-1>.
- Kravchik, M. and Shabtai, A., 2021. Efficient Cyber Attack Detection in Industrial Control Systems Using Lightweight Neural Networks and PCA. *IEEE Transactions on Dependable and Secure Computing*, pp.1–1. <https://doi.org/10.1109/tdsc.2021.3050101>.
- Langner, R., 2011. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE security & privacy*, 9(3), pp.49–51. <https://ieeexplore.ieee.org/abstract/document/5772960/>
- Lee, R.M., Assante, M.J. and Conway, T., 2016. Analysis of the cyber attack on the ukrainian power grid defense use case. *SANS ICS*.
- Li, Z., 2023. Review of PID control design and tuning methods. *Journal of Physics: Conference Series*, 2649(1), pp.012009–012009. <https://doi.org/10.1088/1742-6596/2649/1/012009>.
- Ma, Y.W. and Tsou, C.W. (2024) 'A novel passive-active detection system for false data injection attacks in industrial control systems', *Computers & Security*, 145, 103996. <https://www.sciencedirect.com/science/article/pii/S0167404824003018>
- Marusak, P., Nebeluk, R., Andrzej Wojtulewicz, Cabaj, K., Chaber, P., Maciej Ławryńczuk, Plamowski, S. and Zarzycki, K., 2024. Efficient Cyberattack Detection Methods in Industrial Control Systems. *Sensors*, [online] 24(12), pp.3860–3860. <https://doi.org/10.3390/s24123860>.
- Mokhtari, S. and Yen, K.K., 2024. False data injection attack detection, isolation, and identification in industrial control systems based on machine learning: application in load frequency control. *Electronics*, 13(16), p.3239. <https://www.mdpi.com/2079-9292/13/16/3239>
- Momand, A., Jan, S.U. and Ramzan, N., 2023. A Systematic and Comprehensive Survey of Recent Advances in Intrusion Detection Systems Using Machine Learning: Deep Learning, Datasets, and Attack Taxonomy. *Journal of Sensors*, 2023, pp.1–18. <https://doi.org/10.1155/2023/6048087>.
- Pinto, A., Herrera, L.-C., Donoso, Y. and Gutierrez, J.A., 2023. Survey on Intrusion Detection Systems Based on Machine Learning Techniques for the Protection of Critical Infrastructure. *Sensors*, [online] 23(5), p.2415. <https://doi.org/10.3390/s23052415>.
- Tama, B.A., Lee, S.Y. and Lee, S.C., 2022. A Systematic Mapping Study and Empirical Comparison of Data-Driven Intrusion Detection Techniques in Industrial Control Networks. *Archives of Computational Methods in Engineering*, 29(7), pp.5353–5380. <https://doi.org/10.1007/s11831-022-09767-y>.

Ullah, I. and Mahmoud, Q.H., 2020. A two-level
flow-based anomalous activity detection
system for IoT networks. *Electronics*, 9(3),

p.530. <https://www.mdpi.com/2079-9292/9/3/530>

