
**Bahria University**

Islamabad Campus

Department of Computer Science

# Smart Grid Anomaly Detection System

Final Year Project Report

**Submitted by:** Wareed Ejaz Malik (Roll No. 01-136221-031)
Muhammad Haseeb (Roll No. 01-136221-057)

**Supervisor:** Dr. Saba Mahmood

October 10, 2025

# Certificate

This is to certify that the report titled *Smart Grid Anomaly Detection System* submitted by **Wareed Ejaz Malik (Roll No. 01-136221-031)** and **Muhammad Haseeb (Roll No. 01-136221-057)** in partial fulfillment of the requirements for the degree of **Bachelor of Science in Computer Science** at **Bahria University** is a record of the candidates' own work carried out under my supervision.

**Supervisor:** Dr. Saba Mahmood **Signature:** _____

**Internal Examiner:** Name of Internal Examiner (Title) **Signature:** _____

**External Examiner:** Name of External Examiner (Title) **Signature:** _____

**Project Coordinator:** Dr. Project Coordinator **Signature:** _____

**Head of Department:** Dr. Head of Department **Signature:** _____

**Department Seal:** _____ **Date:** _____

# Acknowledgment

We would like to express our sincere gratitude to our supervisor, Dr. Saba Mahmood, for invaluable guidance and support. We also thank the faculty and staff of Department of Computer Science for the infrastructure and learning environment. Finally, we are grateful to our families and friends for their constant encouragement.

**Wareed Ejaz Malik (Roll No. 01-136221-031)**
**Muhammad Haseeb (Roll No. 01-136221-057)**

# Abstract

This project presents an AI-powered smart grid anomaly detection system with a Flask backend, SQLite database, and a modern web interface. The system analyzes more than 50,000 data points collected at 15-minute intervals and detects normal operation, point anomalies, contextual anomalies, and collective anomalies (5+ consecutive anomalies). The machine learning stack combines Isolation Forest for unsupervised outlier detection, LSTM autoencoders for sequence reconstruction, and a Gradient Boosting classifier as an ensemble decision layer. The application exposes interactive parameter sliders, live computed engineered features, a results page with confidence/probabilities, a history view with exports, and an analytics dashboard. Results indicate high accuracy and low-latency inference suitable for real-world monitoring.

**Keywords:** Smart Grid, Anomaly Detection, Isolation Forest, LSTM Autoencoder, Gradient Boosting, Flask, SQLite.

# Abbreviations

| Abbreviation | Full Form |
|---|---|
| AI | Artificial Intelligence |
| ML | Machine Learning |
| IF | Isolation Forest |
| LSTM | Long Short-Term Memory |
| GB | Gradient Boosting |
| UI | User Interface |
| API | Application Programming Interface |
| SQL | Structured Query Language |
| RDBMS | Relational Database Management System |
| IoT | Internet of Things |
| ROC | Receiver Operating Characteristic |
| AUC | Area Under the Curve |

# Contents


**1 Introduction** **1**

1.1 Background . . . . . 1
1.2 Objectives . . . . . 4
1.3 Scope of Work . . . . . 5
1.4 Report Organization . . . . . 6

**2 Literature Review** **7**

2.1 Introduction . . . . . 7
2.2 Background in Power System Anomaly Detection . . . . . 10
2.3 Statistical and Probabilistic Approaches . . . . . 10
2.4 Machine Learning Approaches . . . . . 11
2.4.1 Support Vector Machines and Kernels . . . . . 11
2.4.2 Tree-Based Algorithms . . . . . 11
2.4.3 Gradient Boosting and Ensemble Learning . . . . . 12
2.4.4 Clustering and Density Estimation . . . . . 12
2.5 Deep Learning and Representation Learning . . . . . 12
2.5.1 Autoencoders and Reconstruction-Error Models . . . . . 12
2.5.2 Recurrent and LSTM Networks . . . . . 13
2.5.3 CNN–LSTM and Attention Mechanisms . . . . . 13
2.5.4 Generative and Adversarial Networks . . . . . 13
2.6 Graph-Based and Spatio-Temporal Frameworks . . . . . 14
2.7 Hybrid and Ensemble Frameworks . . . . . 14
2.8 Hybrid and Ensemble Frameworks . . . . . 14
2.9 Cybersecurity-Oriented Detection . . . . . 15
2.10 Distributed and Edge-Based Detection . . . . . 15
2.11 Comparative Evaluation of Literature . . . . . 15
2.12 Discussion and Synthesis . . . . . 16

**3 Requirement Specifications** **18**

3.1 Existing System . . . . . 18
3.2 Proposed System . . . . . 19

3.2.1 Benefits . . . 20
3.2.2 Solution Areas . . . 20
3.3 Requirement Specifications . . . 20
3.3.1 Purpose . . . 21
3.3.2 User Needs . . . 21
3.3.3 Assumptions and Dependencies . . . 21
3.3.4 Functional Requirements . . . 23
3.3.5 Non-Functional Requirements . . . 23
3.4 Use Cases . . . 23
3.4.1 Use Case Diagram . . . 23
3.4.2 Use Case Description . . . 24
3.4.3 Use Case Description Tables . . . 25

**4 System Architecture & Design** **29**
4.1 System Architecture . . . 29
4.1.1 Context Diagram . . . 31
4.2 High-Level Design . . . 32
4.2.1 Component Diagram . . . 32
4.3 Low-Level Design . . . 33
4.3.1 Activity Diagrams . . . 34
4.4 System Sequence Diagrams . . . 37

**5 Implementation** **41**
5.1 Overview . . . 41
5.2 Development Environment . . . 42
5.3 Data Preparation and Feature Engineering . . . 42
5.3.1 Data Cleaning . . . 42
5.3.2 Feature Engineering . . . 43
5.4 Backend Implementation . . . 43
5.4.1 Collective Anomaly Tracker . . . 44
5.5 Machine Learning Pipeline . . . 45
5.5.1 Isolation Forest Training . . . 45
5.5.2 LSTM Autoencoder Training . . . 45
5.5.3 Gradient Boosting Ensemble . . . 46
5.5.4 Unified Inference . . . 46
5.6 Database and Persistence . . . 47
5.7 Frontend Implementation . . . 47
5.7.1 Signup Page . . . 47
5.7.2 Login Page . . . 48

5.7.3 Dashboard Interface . . . 49
5.7.4 Prediction Result Display . . . 51
5.7.5 Prediction History View . . . 52
5.7.6 Analytics Dashboard . . . 52
5.7.7 Recent Predictions Panel . . . 53
5.7.8 Administrative Monitoring Console . . . 54
5.8 Deployment and Local Server Testing . . . 55
5.9 Performance and Optimization . . . 55

**6 Testing** **56**
6.1 Overview . . . 56
6.2 Testing Objectives . . . 56
6.3 Testing Methodology . . . 57
6.3.1 Testing Environment . . . 57
6.3.2 Test Data . . . 57
6.4 Functional Testing . . . 57
6.4.1 Use Case: User Signup . . . 58
6.4.2 Use Case: User Login . . . 58
6.4.3 Use Case: Prediction Submission . . . 59
6.4.4 Use Case: Prediction History Retrieval . . . 60
6.4.5 Use Case: Analytics Dashboard . . . 61
6.4.6 Use Case: Administrative Monitoring Console . . . 62
6.4.7 Summary of Functional Testing . . . 63
6.5 User Interface (UI) Testing . . . 63
6.5.1 Login and Signup Pages . . . 64
6.5.2 Dashboard Interface Testing . . . 64
6.5.3 Prediction Result Page Testing . . . 65
6.5.4 Prediction History Testing . . . 66
6.5.5 Analytics Dashboard Testing . . . 67
6.5.6 Analytics Dashboard Testing . . . 68
6.5.7 Recent Predictions and Performance Testing of the System . . . 69
6.6 Integration and System Testing . . . 69
6.6.1 Backend–Frontend Synchronization . . . 70
6.6.2 Database Verification . . . 70
6.7 Performance Evaluation . . . 70
6.7.1 Accuracy Metrics . . . 70
6.7.2 Latency and Throughput . . . 70
6.7.3 Scalability and Stability . . . 70
6.8 User Experience Evaluation . . . 70

**7 Conclusion and Future Work** **72**
7.1 Overview . . . 72
7.2 Summary of Achievements . . . 72
7.3 Overview . . . 73
7.4 Technical Insights and Findings . . . 74
7.5 Limitations . . . 74
7.6 Future Enhancements . . . 75
7.6.1 Cloud and Edge Deployment . . . 75
7.6.2 Integration of Federated and Transfer Learning . . . 75
7.6.3 Explainable and Interpretable AI Integration . . . 76
7.6.4 Dynamic Thresholding and Predictive Maintenance . . . 76
7.6.5 Cybersecurity Integration . . . 76
7.6.6 Enhanced Visualization and Mobile Accessibility . . . 76
7.7 Research Implications . . . 76
7.8 Conclusion . . . 77

**A System Implementation and Configuration Details** **78**

**B Extended Test Logs and User Evaluation** **87**

**Appendix B: Full Test Logs and User Evaluation** **87**
B.1 Overview . . . 87
B.2 Functional Execution Logs . . . 87
B.3 User Experience (UX) Evaluation . . . 87
B.4 Error Handling Tests . . . 88
B.5 Performance Summary . . . 89
B.6 Concluding Remarks . . . 89

# List of Figures


3.1 Use Case Diagram of Smart Grid Anomaly Detection System . . . . . 24

4.1 Layered System Architecture of the Smart Grid Anomaly Detection System . . 30
4.2 Context Diagram for the Smart Grid Anomaly Detection System. . . . . 31
4.3 Smart Grid Anomaly Detection System Component Diagram . . . . . 32
4.4 Activity Diagram: Collective Anomaly Tracking . . . . . 35
4.5 Activity Diagram: Analytics Dashboard Workflow . . . . . 36
4.6 Activity Diagram: Anomaly Detection Process . . . . . 37
4.7 Sequence Diagram: Profile Update Process . . . . . 38
4.8 Sequence Diagram: Analytics Dashboard Retrieve Data . . . . . 39
4.9 Sequence Diagram: User Authentication . . . . . 39
4.10 Sequence Diagram: Anomaly Detection . . . . . 40

5.1 User information form and validation hints on signup page. . . . . 48
5.2 Signup page with more fields like department and profiles picture upload feature. 48
5.3 The log-in page for user authentication and secure access to the system. . . . . 49
5.4 Dashboard Interface: displayed parameter sliders, real-time metrics and system-status indicators. . . . . 50
5.5 Extended Dashboard with extra Sliders for parameters and adjusted efficiency calculations. . . . . 50
5.6 Prediction result panel that show the anomaly class, confidence score and probability visualization. . . . . 51
5.7 More examples of predicted results, presenting confidences that vary for different test cases. . . . . 51
5.8 History page rendering stored predictions with advanced filtering and export features. . . . . 52
5.9 Anomaly: Dashboard as an overview of trends, statistics and renewable integration. . . . . 53
5.10 Extended Analytics view showing comparison of anomaly frequencies and daily confidence levels. . . . . 53
5.11 Panel of the dashboard providing recent predictions with timestamps along with related confidence scores. . . . . 54

5.12 Admin console for health and associated features, API latencies, as well as anomaly counters. . . . . 54

6.1 Login and Signup Pages—user registration & authentication testing . . . . . 64
6.2 Dashboard Interface—initial parameter input testing . . . . . 65
6.3 Dashboard Interface—real-time feedback verification . . . . . 65
6.4 Prediction Result View—Normal operation . . . . . 66
6.5 Prediction Result—Point anomaly detection case . . . . . 66
6.6 Prediction History Page – Overview of most recent analysis activity . . . . . 67
6.7 Analytics Dashboard – anomaly frequency and renewable ratios . . . . . 67
6.8 Prediction History Page – Overview of most recent analysis activity . . . . . 68
6.9 Analytics Dashboard – anomaly frequency and renewable ratios . . . . . 68
6.10 Extended-AD: anomaly distribution and confidence statistics . . . . . 69
6.11 In Summary, Predictions Analysis and Recent System Performance . . . . . 69

A.1 Dashboard interface during live testing of an anomaly. . . . . 84
A.2 Prediction history view with detection confidence and types displayed. . . . . 84
A.3 Analytics report in concise view representing anomaly disposition and risk level. 85

# List of Tables


2.1 Sample of Smart-Grid Anomaly Detection Studies . . . . . 16

3.1 Use Case UC-01: User Authentication . . . . . 25
3.2 Use Case UC-02: Execute Anomaly Detection . . . . . 26
3.3 Use Case UC-03: Visualization and Monitoring . . . . . 26
3.4 Use Case UC-04: Configuration Management . . . . . 27
3.5 Use Case UC-05: Generierung und Export von Berichten . . . . . 27
3.6 Use Case UC-06: User Management . . . . . 28
3.7 UseCase UC-07: HealthMonitoring . . . . . 28

4.1 Component Descriptions . . . . . 33

6.1 Functional Test Cases regarding User Signup . . . . . 58
6.2 Functional Test Cases for User Login . . . . . 59
6.3 Prediction Submission Functional Test Cases . . . . . 60
6.4 Functional Test Cases of Prediction History Retrieval . . . . . 61
6.5 Functional Test Suites for Analytics Dashboard . . . . . 62
6.6 FTC for Administrative Monitoring Console . . . . . 63

B.1 Summary of the User-Experience Survey (Likert Scale 1—5) . . . . . 88
B.2 UI Error Handling Test Cases . . . . . 89

# Chapter 1

# Introduction

## 1.1 Background

The modern electricity grid is being radically changed. In both technology and form, this evolution - driven by infusions of renewable energy, ubiquitous sensing technologies and real-time communication - has spawned what is now commonly known as the smart grid, but the smart grid is not just a more sophisticated grid, it's also a complex cyber-physical system where physical infrastructure and digital intelligence are tightly coupled. Through the use of distributed generators, smart-meters, phasor measurement units (PMUs), SCADA or other intelligent devices grid operators have access to high-resolution operational data as never before. The introduction of operational data allows for fine-grained monitoring and predictive control, but at the same time also raises new challenges with regard to data quality, system reliability and error detection.

The overall effect of the distributed generation (DG) using renewable sources such as solar PV and wind turbines is to introduce a natural stochastic element in the grid behavior. In contrast to conventional generation capacity, which may be close moderated, wind power is variable depending upon natural factors of wind flow and sunlight intensity or temperature and so may not give rise to dispatch costs. It's this variability that creates variation throughout the network, impacting on voltage stability, reactive power balance and load distribution wherever it migrates. Furthermore, the customer-side expansion of the use of electric vehicles, battery energy storage and demand response schemes has made load profiling more uncertain. The classic straight-line flow from generator to load has become a new circular puzzle of sorts regarding direction, and power - and regarding place on the grid in which producers (entities that both produce power and consume power) move fluidly. So as a result its been hard to find examples of things going bad or have some new destabilization factor in a nonlinear but very strung out If that could even make sense. The historical trends of the condition diversion choice are collected most do not even refine the former bureaucratic formulae for their trend (monitoring with convention-based technique) Turbine run When out standard alarm, and parameter away data back to an error

by a few more tens of kilometers Points exceed committee within go alarm points will return. So in an application of this nature that is generic (not near nominal voltage), if you're not close to nominal voltage (5What seems to be a prototype developed by Microsoft Research that solves what? Now that we have some modern methods of measuring power system error (read: system failure) out of the way, let's see what the new and improved techniques are. To overcome these drawbacks, the methods of artificial intelligence (AI) and machine learning (ML) have brought new tools for detection of anomalies. Unlike the fixed threshold based rules used so far, learning models come to be able to learn and analyse statistical regularities in sentations of huge quantity of historical data by computational means. They are able to manage non-linear relation between voltage, current, temperature and demand naturally, with a more flexible ability of detecting the abnormal operating condition. Whereas unsupervised methods, such as k-means clustering [22], auto-encoders [23], and diffusion maps [24] are able to learn the patterns from data, without having to be shown examples of defective or anomalous behavior, su- supervised algorithms, such as SVMs [25], random forests [26] and gradient boosting decision trees (GBDT) [27-29] have traditionally been extremely sensitive and require labeled datasets in order to establish operating conditions that are either normal or anomalous with high sensitivity. In particular, deep learning has become a powerful tool to tackle temporal and spatial patterns of smart grid data. As seen by the experts, there are recurrent neural networks (RNNs) and long short-term memory (LSTM) architectures, which pick up long-range temporal dependencies. They are thus able to identify context anomalies that are caused on a time horizon that is extended. The convolutional neural networks (CNNs) are an effective method for feature extraction of multidimensional sensor data to get the spatial features representing the interactions between distributed nodes. Recent hybrid models of CNN and LSTMs architectures can treat temporal sequence and spatial correlations simultaneously. This is an important skill in which anomalies are transported from interconnected components to others. For instance, a phenomenon of cascading voltage collapse through local transformer overload may occur simultaneously at the substations within a limited distance even though they are not visible at all when spatial-temporal dependence is not made explicit. Another aspect that is important in current research on anomaly detection is to introduce attention mechanisms and graph-based models. By only paying attention to influential components of the time series data or the sensor readings, attention-based networks increase both the accuracy of the detection and the interpretability of the detection. On the other hand, graph neural networks (GNNs) make use of the inherent graph structure of power systems, and use this to explicitly model relational dependencies. For instance, they enable the system to learn about topological abnormalities such as line faults or abnormal flows. A purely sequential model is not able to capture such occurances.

The design of GNNs and temporal encoders, which are also known as spatio-temporal graph networks, are a frontier to the intelligent grid monitoring systems.

As the complexity of grid data increases, the importance of explainability and interpretability

also increases.Black Box: Despite being impossible to understand, deep learning models are able to achieve astonishingly good results. Opacity is a major impediment to their use in large applications, however, such as energy infrastructure. Decision makers, both operators and regulators, need high-quality insights for their decisions (and in an environment where decisions such as anomaly alerts could result in costly policies). An AI framework to explain the things (and interpretable) is now being incorporated into more and more anomaly detection systems. That way, when there are exceptions, that end users can understand why those exceptions occurred - they couldn't find sensors or there was a fault in the equipment. SHAP values, layer-wise relevance propagation, attention heatmaps and other tricks are used to try to address the gap between model performance and human trust. In smart grids, when the model structure has been decided there is a whole set of practical deployment and anomaly detection problems that are to deal with. Some of them are related to the data quality and scalability and latency. Due to the failure of sensors, noise or environmental interference, the data streams from grid are regularly incomplete and noisy, as well as asynchronous. Modeling gaps in the field can directly cause a model to be unreliable, unless these are accounted for through the robust downstream pre-processing pipelines (e.g. through imputation, smoothing and normalization). However, an inference time is needed to make inferences and we need a light-weight architecture that runs in real time (no accuracy loss) on edge devices or on local servers. Edge computing infrastructures are starting to make their presence and pairing containerized deployments with these infrastructures using runtimes like Docker and Flask puts anomaly detection in distributed systems in a race for scalable and low latency systems.

But then, speaking from another aspect of cybersecurity, anomaly detection is more needed now than ever. The digitization of power systems has two sides: it offers an expanded theater of cyber crime on the one hand, while at the same time it creates new weaknesses in the communication and control protocols on the other. Adversarial attackers can either change the readings of sensors or create fake data streams to interfere with the performance of real systems and interrupt their service. In such security means, it is possible the learning normal patterns not working suddenly in turning the defense crisis at some area and define-based intrusion detection systems bypassing or cannot be used. All this complicates further the troubleshooting process. Last but not least, taking into account that smart grids are getting more decentralized (i.e., moving into the microgrid or even to agent-based trading area) anomalies have also to deal with decentralized data structures. The concept of federated learning is put forward as one possible solution to this issue where multiple local sites (e.g., substations or microgrids) collaboratively train their models without exchanging their data minus the bare input points. This is by design to protect privacy and at the same time maintain some level of overall situational awareness. It does not, also, repeat the tax payer funded lock-in of previous power production. If some day you have a lot of data, cloud-based deployments allow for cheap training and unified visualization via APIs for use with your existing SCADA system.

Despite significant advancements in smart grid technologies, current monitoring and anomaly

detection mechanisms are severely limited in their ability to be deployed in a large scale for real life operational systems. Existing approaches also have three major faults: 1. Limited coverage of abnormalities. Most of them only capture point anomalies, which means that they will be able to detect individual data points that are much different than what is anticipated. They fail to detect more subtle types of anomalies such as context anomalies (which look completely kosher in isolation, but which aren't kosher in special circumstances: for example ones that are of concern to a utility operator on deep night because they represent high load during a heat wave) or collective anomalies (where a group of correlated data points collectively shows abnormal behavior even though all the individual ones considered in isolation are benign). 2. Poor operator interactivity. Most do not have a user friendly real-time visualisation interface, intuitive dashboards for users or clear, actionable insights in a format that isn't endemic for current detection reports with this effect that threat response teams aren't able to make judgement calls even in times of high pressure - this causes operational decisions to be delayed as mentioned previously. 3. Not having enough scalability and security However, the existing software stacks often fall short on a number of enterprise-grade security and scalability and deployability factors. In many cases, they do not have robust RBAC (role-based access control), secure authentication with single sign-on and loosely coupled/modular architecture that would allow scalable integration into the production environment for a utility. The objectives of the project is to fill-up these gaps with the implementation of a full anomaly detection system for web, with AI and ML. The idea is to aggregate multiple independent machine learning models, which work in conjunction with a responsive secure frontend visualization - giving users a continuous view on the current health of all the types of anomalies in the system.

## 1.2 Objectives

The key objectives of the project are as follows:

- install and implement an extensive anomaly detection engine (software); able to correctly classify 4 classes of anomalies; i.e. normal of the system (baseline), point, contextual & collective anomalies from both real smart grid data and any one of several simulated data sets available to us

  The systems must accept, process and analyze more than 50,000 data inputs every fifteen minutes, and remain equally sensitive in the detection capability and response time to meet industry's requirements for real-time operations.

- To Develop the website A resilient web application that is scalable with secure features that are built with Flask micro frameworks and SQLite database. Implement RBAC to differentiate roles of users such as admins, analysts and field operators. This will ensure the integrity of the data and the entire system.

- To dynamic and attractive charts showing anomalies and system health in real-time. These charts are a form of making the situation immediately visible for operators, who will be in the right place to make fast decisions.

- To Manage user comprehensive management capability: customisable profile setting, history stores who checked records, what, from which site, preventing anomalies from tracking between, anomalies held on different systems; to export detection result files and system log data in several formats (e.g. csvfile - json - pdf) would be essential features for reporting purposes. Because natures we will provide hard-copies of these translations translation do not mean that employees should only do this but should be able to complete this task in their language (despite the availability of translations) if that pdf file is referenced for any translation it does not.

## 1.3 Scope of Work

The project covers the whole process life cycle of a smart grid monitoring AI based anomaly detection system. The backend infrastructure is based on a Python/Flask framework with an SQLite engine, which is easy to set up and manage while being robust enough to store user credentials, system configurations (anomaly logs and transformed telemetry data) etc. In the pre-processing and feature engineering steps, various industry standard Python libraries like Numpy, Pandas etc are utilized to prepare the software for quickly processing industrial time series large scale datasets.

The approach of using a system for anomaly detection is a hybrid of machine learning and deep learning. Just to name a few, LSTM Autoencoder learns relationships between data other than time stamps, Isolation Forest is a supervisory-free volume anomaly detector, and Gradient Boosting based ensemble testing (XGBoost) is used as a Contextual Anomaly Detection for multi-level errors, which provides more overall coverage for types of anomaly events.

The system is tracking limited primary numbers of the smart grid: all 13 listed below, voltage level demand and current flowing through wires, some other primary numbers: metering sensor readings renewable energy generation (wind/solar) output frequency voltage stability temperature humidity solar insolation engineered/clustered its features - far from just indices within the network it may have computed for it; could be things like efficiency ratios within a network, penetration levels for renewables in a region, etc. that make sense to understand the model back end not only on grounds of having less memory/compute space but also as having clearer commit interpreted!

User Interface - The user interface is modern and by current web standards, it is structured using HTML5 and styled using CSS3 with the assistance of Bootstrap 5 for a responsive (mobile friendly) page layout. Interactive visualisations come from Chart. js, that not only provides you with real-time updates but also allows you to zoom in a chart or to filter out some data

points, very appropriate for such drill-down behaviors. The current implementation has been validated in data from a high-fidelity simulation of a real-world smart grid that is in the form of an easy-to-use standalone web app but there is no intention whatsoever to use it for any sort of production deployment. Most importantly, it does not connect to real-world IOT sensors or SCADA systems (although that area is an interesting one for future development). It cannot even offer more advanced types of clouds like containers or auto-scaling, let alone container-focused service providers. The system does not have the possibility of providing all of these. These will be promising lines to explore in the future.

In the rest of this thesis, it is intended to outline a logical and detailed pathway through the conception of the project through implementation and design to evaluation.

## 1.4 Report Organization

The remainder of this thesis aims at providing a holistic and rational pathway through the conceptualization, design and implementation, and evaluation of the project. In Chapter 2, a review of the published literature on the technology of ANOMALY DETECTION technologies at SMART GRIDS, and the latest AI/ML Datasets are disclosed. A comparison between common commercial practice or academic case studies is carried out on a non-statistical basis. Chapter 3 presents system requirements, which are categorized into functional (e.g., user authentication, automatic patterns recognition and data ingestion) and non-functional requirements (e.g., Efficiency). The former again have to be supported by the tender promises, which we have already kept in our company for several years. Chapter 4 involves the full architecture of the system. From the system Composition Diagrams to Data Flow Charts; from particularization of the functions of service domains that can be executed on various nodes of the system, to the description of the behavior of the models with use cases and sequence diagrams (more detailed at a structural and behavioral level). Chapter 5 is the action; it's the doing of such-and-such experience, the sidebar about how we did back-end interface design right but it took months instead of minutes, because pipelining machine learning, and security around real-time data processing and viewing because who has time for just-in-time merchandising on-line? Item 6. Assess the performance of the system using quantitative measures such as: recall/precision/F1 score, Qualitative quotes from user testing and a comparison with other existing techniques (such as: speed, when something had to be written to disk).

# Chapter 2

# Literature Review

## 2.1 Introduction

The evolution of the power grid to become a cyber-physical systems (CPS) network has transformed the structure and operation of power grids. Recent advances in sensing, communication and computation technology has made possible what we call as a "smart grid" - distributed decentralized adaptive information rich architecture that seamlessly combines traditional electrical networks with state-of-the-art IT. Indeed, smart grids are not only the upgrading of conventional power systems; they represent a new ideology for energy management where renewable distributed generation and electric vehicles (EV), distributed storage systems and demand response loads would become part of one unique interactive system. The purpose of the evolution is to increase reliability, efficiency, durability and environmental performance. But the very interconnectedness that makes these systems so powerful also makes it more complex and prone to failure and the potential for operational malfunctions that can add risk to grid stability and security.

Anomaly detection is a prerequisite work for Smart Grid analytics for such a context. The large volume of sensors, smart meter and supervisory control and data acquisition (SCADA) system data offers an opportunity for real-time monitoring while also raising a knowledge discovery problem on massive amounts of changing information that is typically noisy, incomplete w.r.t. more importantly transient in nature. For all these large patterns it is valuable to detect patterns which are not according the general pattern. These oscillations can have different sources from aging equipment or disturbances to a malicious cyber attack and an unanticipated load surge increment. A key benefit of early warning detection of these anomalies is that the control room operators can take appropriate action prior to the progression of small vibration into cascading failure and widespread grid blackouts. Early anomaly detection, from an economic perspective, can lead to significant saving in the operating cost through avoiding unplanned outages and optimising the maintenance schedule, but from security perspective, it will translate into the first line of defence against sophisticated cyber-physical attacks.

It is, therefore, the task to secure smart grid operation from a reliability and safety point of view which makes the field of anomaly detection (outlier detection or novelty detection) crucial [1]. Conceptually Anomaly detection is the process of identifying members or patterns in the data which do not follow an expected behavior according to a set of routine examples. From a statistical point of view, if $x_i \in \mathbb{R}^d$ are randomly drawn from some multivariate probability distribution $p(x)$ then an anomaly occurs if the condition

$$A(x_i) = \begin{cases} 1, & \text{if } p(x_i) < \theta, \\ 0, & \text{otherwise,} \end{cases} \tag{2.1}$$

where $\theta$ is a statistical tolerance level for rarity based on observed data or theoretical model. But the way probability theory is defined is, too beautiful for practical direct use. Real-world smart grid data is hardly clean, nicely probability distribution and non-stationary, high-dimensional, noisy and incomplete in nature. On the other hand, the inter-relation between variables (e.g., voltage and current) are non-linear as well as time varying which does not allow use of standard model based assumptions.

The operational anomadetection requirements that smart grids need to meet are characterized by the considering of high sensitivity and high specificity as necessary; i.e., real anomalies need to be detected with minimum – if any at all – false alarms, which would add un-necessary stress on human operators. Formation of such detection systems requires in depth understanding of temporal and spatial power network dynamics. In the literature, conventional statistical techniques (e.g., Gaussian mixture models, auto-regressive integrated moving average (ARIMA) and control chart methods) belonging to a first generation of solutions can be found for anomaly detection in power systems. These methods were based on the assumption of stationarity that, as the data accumulated through time, the data flexed less (or took more predictable forms) as events unfolded over time. Even if able to perform well in stationary environment, such techniques would not stand up the random and time varying load profile of modern grid where generation source distribution shifts randomly depending from weather conditions or user habits.

The second of these great tipping points would be the introduction of machine-learned approaches. Approaches like k-nearest neighbours, support vector machine or clustering made data-driven modelling possible that could describe more complex dependencies without the need of physical modelling. Machine learning had stood us in good stead to fall back on what it had seen before, and come up with its own general response free of any artificial bias. Yet, these approaches have their limitations too: The in dependence-on-features extraction to which the former relies and the difficulties to represent temporal-long dependencies (crucial for power system behavior) to which the latter struggles. For example, a voltage drop may only be deemed anomalous if it is experienced in the context of a temporal pattern that is not experienced elsewhere and this may not easily be viewed within static models.

The next revolution was Deep learning on anomaly detection in smart grid. Feature-laden

structures like autoencoders, convolution network (CNNs) and recurrent neural network (RNN) including Long Short-Term Memory (LSTM) may have driving forcing to brilliant revolution in regard to time series modeling: Learning representation that is abstract aspect of raw data itself. In particular, autoencoders based on LSTMs are becoming increasingly popular since they are able to reconstruct temporal sequences and use reconstruction errors as proxy to anomalousness. In such approaches normal dynamics of grid is transferred to the model and if there is a deviate then its shows signs of anomaly. It has been especially successful for the detection of (contextual and collective) anomalies: in such types of anomaly, the abnormality is represented by some combination of values or statistics, or it only appears to be abnormal when compared to other entities. Deep learning based approaches are also attractive for their scalability and it can easily deal with multimodal data streams or fusions (e.g., electrical, environment, operation variables) under the same framework.

But there are still some ways in which deep learning models struggle. They are a black box most of the time and there is no interpretability for human, besides it takes heavy computation resource cost in training stage and deployment. Moreover, the dependence on large labelled training sets may not prove applicable to certain domains in which there are very limited or expensive to acquire anomaly label data. These limitations have recently boosted the interest for hybrid models, which combine unsupervised learning algorithms (as Isolation Forest, or other statistical outliers addressing approaches) with a deep neural architecture that measures long term dependencies- In this hybrid paradigm unsupervised algorithms are used to create potential anomalies and then deep models can be used to filter the detections and provide context around them making the produced results much higher quality.

Here, there is a new frontier in learning with graphs/explainable AI. The reason that we are using GNNs is that power grids are typically represented as graph structured topologies, including nodes (buses or/and substations) and edges (transmission lines). GNN-based models are able to identify a local fault spreading to other nearby parts as they basically learn node-level and networkwise representations. Meanwhile, interpretable AI techniques aim to make sure that the results of machine learning can be understood, so that operators know why some events have been identified as anomalies. For critical infrastructure such as energy systems, accountability and trust is important and this interpretability is a significant advantage.

Overall, the research roadmap of smart grid anomaly detection is on its way to leave behind a static rule-based reasoning direction and move to be driven by dynamic data and hybridized intelligence. With the complexity of today's grids, there is however no better indication for us to push for systems that have to not only be accurate, but also adaptive (i.e. have the ability to learn and continuously adapt as they gain more knowledge), interpretable (meaning humans should be able to understand how decisions were made by algorithms and stand a fair chance at challenging them) and scalable. The next-gen anomaly detection will take advantage of advances in distributed computing, federated learning and cloud-edge synergy to enable real-time decision making without compromising the privacy of enterprise data. In this perspective,

anomaly detection has been transformed from a statistical aberration into a smart grid resiliency Lego - the mechanism needed to build a stable, efficient and sustainable power system that underpins today's civil society.

## 2.2 Background in Power System Anomaly Detection

Historical power-system monitoring was based on deterministic and statistical methods with control theory as a motivation. Initially, these applications were focused on fault detection through the use of residual-based analysis where the difference between measured and model predicted outputs was monitored. If the residual $r_t = y_t - \hat{y}_t$ was greater than a predefined threshold it generated a fault signal. This presentation has been done under the assumptions of linearity and Gaussian noise.

A typical model for temporal modeling was the Auto-Regressive Integrated Moving Average (ARIMA) process:

$$x_t = \sum_{i=1}^{p} \phi_i x_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t, \tag{2.2}$$

where $\epsilon_t$ represents white noise. ARIMA, however, was able to capture only short-term correlations and needed the signal to be stationary with less adaptability into non-linear trends of grid operation [2].

Control-chart procedures such as Shewhart and CUSUM charted the mean or variance of measurements to identify shifts that were defined in terms of either

$$S_t = \max(0, S_{t-1} + (x_t - \mu_0 - k)), (2) \tag{2.3}$$

where and $k$ specifies sensitivity. These approaches are computationally efficient, yet assume well-defined nominal distributions that are seldom met in practice grids.

## 2.3 Statistical and Probabilistic Approaches

However with increasing data volume, it was no longer practical to have only deterministic analysis of residual behaviour. Probabilistic models such as GMMs and Bayesian networks were able to model uncertainty in a more richer manner. The data distribution is modeled using a GMM in the use case of oath.

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k), \tag{2.4}$$

where $\pi_k$ and $\mu_k$ are the mixture weights, mean vectors and $\Sigma_k$ are the covariance matrices. Data points in the low density region will be treated as anomalies if $p(x_i)\tau$. Although suitable to

linear dependencies, the limitation of PCA is its inability to describe nonlinear relations between current, voltage and power factor. The Kernel-PCA and Independent Component Analysis (ICA) were work around nonlinear extension but its parameter tuning are cumbersome. Sequential aberrance detection was also utilized by Hidden Markov models (HMMs). The state sequence $S = \{s_t\}$ is used to represent discrete operational states of the grid. The probability $P(O|S, \lambda)$ of an observation sequence O was compared to a baseline. An anomaly is indicated by an abrupt decrease in the likelihood. However, these probabilistic sequence models were constrained by the Markov assumption and could not learn long-range correlations.

## 2.4 Machine Learning Approaches

With the deployment of smart meters and PMUs, data acquired became not only big and complex, but also rich in nonlinear patterns. There were data-driven options with the use of supervised and unsupervised machine-learning methods.

### 2.4.1 Support Vector Machines and Kernels

Support Vector Machines (SVMs) were among the earliest learning based detectors on grid anomalies. One-Class SVM draws a decision boundary in high-dimensional feature space to separate the nor data from the origin: 311 312 313 For instance, if we have two-dimensional input feature vectors, denoted by x =(x 1.

$$\min_{\mathbf{w},\rho,\xi_i} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu n}\sum_i \xi_i - \rho, \tag{2.5}$$

such that $(\mathbf{w} \cdot \phi(x_i)) \geq \rho - \xi_i$ where $\xi_i \geq 0$. The nonlinearity in the decision boundaries is obtained through the kernel function $\phi(\cdot)$. Though theoretically well founded, training complexity $O(n^3)$ is a barrier to real-time scalability.

### 2.4.2 Tree-Based Algorithms

Ensemble trees provided an increase in scalability. The computation by the Isolation Forest (iForest) algorithm [3] is separated into a tier based on recursive partitioning of anomalies. The mean distance $E[h(x)]$ is thus a measure for the average speed of isolatedness, normalised with respect to $c(n)$:

$$s(x, n) = 2^{-\frac{E[h(x)]}{c(n)}}, \quad c(n) = 2H(n-1) - \frac{2(n-1)}{n}, \tag{2.6}$$

where $H(\cdot)$ is the harmonic number. Since the anomalies are "few and different," they need less splitting. Due to its linear time complexity, iForest is widely used in real-time PMU monitoring.

Derivatives like Extended iForest and Streaming iForest decrease the memory overhead even further.

### 2.4.3 Gradient Boosting and Ensemble Learning

Gradient Boosting Machines (GBM) [4] aggregate weak decisions trees in a forward stage wise fashion to minimize the loss:

$$F_m(x) = F_{m-1}(x) + \nu\gamma_m h_m(x), \tag{2.7}$$

where the learning constant $\nu$. XGBoost [5] and LightGBM [6] further refined this approach with regularization, histogram-based splits and GPU acceleration. Their feature-importance visualization facilitates the interpretability for power-grid management.

### 2.4.4 Clustering and Density Estimation

Unsupervised approaches as k-means, DBSCAN and Local Outlier Factor (LOF) are able to detect anomalies in an unlabelled way. LOF computes local density deviation:

$$\mathrm{LOF}k(x_i) = \frac{1}{|N_k(x_i)|}\sum x_j \in N_k(x_i)\frac{\mathrm{lrd}(x_j)}{\mathrm{lrd}(x_i)}, \tag{2.8}$$

where lrd is the local reachability density. LOF can have great ability for capturing the local anomalies, but is sensitive to $k$ and cannot consider time. Altogether, although classical machine learning models provided higher scalability and automation levels than CSE analysis algorithms, they did not consider time-solved dependencies crucial to electrical stability assessment.

## 2.5 Deep Learning and Representation Learning

Deep learning techniques marked a turning point, according focus on for hierarchical feature abstraction from raw signal. Neural Models – feedforward, recurrent and convolutional – are the new emperors in the landscape of anomaly detection.

### 2.5.1 Autoencoders and Reconstruction-Error Models

Autoencoders find a way to compress and decompress normal behavior and raise an alert if the reconstruction error is huge:

$$\mathcal{L}AE = \frac{1}{N}\sum i = 1^N \|x_i - \hat{x}_i\|_2^2 . \tag{2.9}$$

When $\mathcal{L}_{AE} > \tau$ the instance is anomalous. Types include Denoising Autoencoders (coping with noise) and Sparse Autoencoders (with $\ell_1$ penalty). Malhotra et al. [7] used LSTM based encoder–decoders for multi-sensor anomaly detection with early warning of load changes.

### 2.5.2 Recurrent and LSTM Networks

Long Short-Term Memory (LSTM) networks proposed by Hochreiter and Schmidhuber [8] address the vanishing-gradient problem in RNNs through gated memory cells:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f), \tag{2.10}$$

$$i_t = \sigma(W_i h_{t-1} + V_i x_t + b_i), \tag{2.11}$$

$$\tilde{C}t = \tanh(W_c[ht - 1, x_t] + b_c), \tag{2.12}$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t, \tag{2.13}$$

$$h_t = o_t * \tanh(C_t), \tag{2.14}$$

where $f_t, i_t, o_t$ are forget, input and output gate. For the smart-grid: LSTMs for long-term consumption behaviors and cyclical load patterns.

### 2.5.3 CNN–LSTM and Attention Mechanisms

Hybrid CNN–LSTM architectures model the spatial and temporal dependencies in the data. CNN layers lend the capability to get local correlation (e.g., neighboring sensors), while LSTMs can catch dynamical information along time. Moreover, models equipped with attention mechanisms can be interpreted in more detail as they assign weights $\alpha_t$ to timestamps:

$$\alpha_t = \frac{\exp(e_t)}{\sum_k \exp(e_k)}, \qquad e_i = v^\top \tanh(W_h h_t + b). \tag{2.15}$$

Guo et al. attention-based LSTM autoencoders were able to obtain 10–15 % higher F1-scores on renewable-integrated grids[9]).

### 2.5.4 Generative and Adversarial Networks

Another approach to anomaly detection is generative adversarial network (GAN), which can model the distribution of data implicitly. The generator $G(z)$ generates fake samples, and the discriminator $D(x)$ distinguishes real from fake:

$$\min_G \max_D \; \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_z}[\log(1 - D(G(z)))]. \tag{2.16}$$

Anomalies are detected when reconstruction using $G$ is not possible, or the confidence of discriminator goes down. GAN-based detectors achieve good performance for highly multimodal data with complex structures, but are computationally expensive and hard to stabilize.

## 2.6 Graph-Based and Spatio-Temporal Frameworks

Power networks are naturally represented by graphs, which nodes and edges correspond to buses and transmission lines, respectively. Graph Neural Networks (GNNs) exploit this structure for the purpose of detecting faults that are spatially correlated. A generic GNN update rule is

$$h_v^{(l+1)} = \sigma\left(W^{(l)} h_v^{(l)} + \sum_{u \in \mathcal{N}(v)} \frac{1}{|\mathcal{N}(v)|} W_N^{(l)} h_u^{(l)}\right). \tag{2.17}$$

Donnot et al. [10] used GNNs for the prediction of post-fault line flows and detection of anomalies, which are deviated from dynamic thresholds. Chen et al. [11] combined GNN and LSTM to model topology correlation and temporal dependency, and achieved high accuracy on IEEE-118-bus benchmarks.

## 2.7 Hybrid and Ensemble Frameworks

A recent literature focuses on hybridization- joining together models that complement each other due to their efficiency and robustness. Chalapathy and Chawla [12] categorised mixed frameworks into cascaded, parallel and hierarchical ensembles. The (general) hybrid score can be expressed as a where the quantities "a" are associated with arm contributions, and the quanitities "b" compete them.

$$S(x) = \sum_{i=1}^{M} w_i s_i(x), \tag{2.18}$$

where $s_i(x)$ is a normalized score of outlier of the $i$th base model amongst Anomaly Scores from $M$ individual models. Weights $w_i$ can be either fixed or meta-optimized. For example, iForest could yield an unsupervised score that temporally refines the LSTM autoencoder and is then classified by a GBM. Hybrid approaches provide more robustness towards missing data and offer adaptation to multiple types of anomalies.

## 2.8 Hybrid and Ensemble Frameworks

Recent work focuses on hybrid- ization — combining complementary models in order to take advantage of a variety of different strong-points. Chalapathy and Chawla [12] had categorised hybrid approaches in cascaded, parallel and hierarchical models. The hybrid score in the most

general form is expressed as

$$S(x) = \sum_{i=1}^{M} w_i s_i(x), \tag{2.19}$$

where $s_i(x)$ is a normalised anomaly score from M member models. The weights $w_i$ can either be fixed or learned through meta-optimization. For example, iForest may give an initial unsupervised score refined temporally through a LSTM autoencoder and classified directly by a GBM. The hybrid system is more robust against missing values and to detect different types of anomalies in various dimensions.

## 2.9 Cybersecurity-Oriented Detection

False-data injection (FDI) attacks pose a security threat to smart grids. Detection projects typically couple machine learning and physical-law constraints. A residual-based detector computes

$$r = z - H\hat{x}, \tag{2.20}$$

where $z$ is the measurement vector and $H$ is the Jacobian of the power-flow model. An FDI attack modifies $z$ to set $r = 0$ and thus escapes from classical detectors. Hybrid PINNs bring sensitivity back through the enforcement of Kirchhoff constraints within neural architectures.

## 2.10 Distributed and Edge-Based Detection

Recentralized anomaly detection requires high network bandwidth and has latency. Edge computing alleviates this by performing light-weight detection near data sources. Throughout the training process, federated learning allows joint training of models without sharing raw data and hence protecting privacy. The global model update at round $t$ can be written as

$$w^{(t+1)} = \sum_{k=1}^{K} \frac{n_k}{n_{tot}} w_k^{(t)}, \tag{2.21}$$

where $w_k^{(t)}$ are local parameters. Edge-based architectures are promising for the IoT-scale national-grid monitoring [13].

## 2.11 Comparative Evaluation of Literature

In Table 2.1, we compare several major approaches and their representative datasets and results. It does fit A4 page by squeezing text width, still achieve readablity.

Table 2.1: Sample of Smart-Grid Anomaly Detection Studies

| **Approach** | **Studies Included** | **Pros** | **Cons** |
|---|---|---|---|
| ARIMA/PCA | [14, 2] | Interpretable, low computational burden. | Poor generalisability over nonlinear, seasonal data. |
| iForest / SVM | [3, 15] | Effective, model-free; can deal with high-dimensions. | Not aware of the along with instantaneous relationship, no semantic reasoning. |
| LSTM/Autoencoder | [7, 9] | Elucidates sequential dependency; insensitivity to noise. | Training costly; large datasets needed. |
| CNN–LSTM / Attention | | [16, 11] Joint spatial–temporal modeling; interpretable. | Computationally heavy; tuning sensitive. |
| GNN/GCN–LSTM & [10] | Uses a grid topology; identifies cascading failures. | Requires full network information; scaling non-trivial. | Needs complete network data; scaling difficult. |
| Hybrid (iForest + LSTM + GBM) | [12, 17] | High accuracy; multi-modal; explainable through GBM. | Complex integration; latency in the context of real-time. |

## 2.12 Discussion and Synthesis

Over the decades of development, several trends in literature are observed as:

**1. Methodological Convergence.** The boundaries between statistical, machinelarning and deep-learning lines are coming together in hybrid frameworks. Ensemble detectors can trade off interpretability and accuracy, which may be desirable in safety-critical systems.

**2. Temporal and Spatial Awareness.** Early solutions have disregarded time; newer alternatives focuses on sequential and graph structured models. The current frontier combines temporal LSTMs with spatial GNNs [46, 24].

**3. Explainability and Human-Centric Design.** Today deployments value user interpretability and decision support, over black-box accuracy. Feature attribution techniques and visual dashboards are used to translate model predictions into actionable insights.

**4. Practical Constraints.** The majority of the models are laboratory prototypes which are not scalable in field. Some of the such challenges are data loss, communication delay, system

security and privacy as well as government regulation.

**5. Research Gaps.** However, despite the progress, standard baselines, common evaluation measures and real-time deployment are yet scarce. Detection, visualization and access control are few of the features which have been combined in a single system so far.

These observations provide a rationale for the hybrid web-deployable system proposed in this thesis where statistical isolation, temporal learning and supervised refinement have all been integrated into an operational platform capable of performing real-time anomaly detection on smart grids.

# Chapter 3

# Requirement Specifications

## 3.1 Existing System

Modern energy infrastructures have a digitalised power industry through intelligent monitoring, predictive analytics and automated control. Despite these technological advances, most of the currently used tools are still restricted by their detection method. Conventional grid management systems were optimized to be used in fixed network conditions where generation had predictable schedules, and load was constant. This method of operation was previously assumed but with the introduction of renewable resources (DG) and dynamic tariffs or pricing schemes, these assumptions were violated. Thus far, the systems developed do not consider the complex non-linear behaviour in smart grids that are applied in practice for some simulation [8]-[12].

Traditional fault detection systems are mostly rule-based or threshold-based. These techniques compare the incoming sensed data with prior operational ranges for such dynamic functions as voltage, current, and load. Alert A warning exists that is thrown when a measured result is outside the specified limits. Ease of application aside, these approaches are reactive as opposed to predictive and cannot model contextual or sequential dependencies between variables. For example, a temporary increase in the voltage due to intermittency in renewables may be regarded as an anomaly but normal behaviour when a particular scenario exists in the grid. This results in false alarm, operator fatigue etc. On the other hand, small differences, which accumulates over time, may not be discovered until they become expensive failures.

Several statistical techniques were used to increase the precision of anomaly detection. Kalman filters, autoregressive models, and moving average smoothing methods are some of the popular methods. However, the assumptions of linearity and stationary distributions of data are not generally applicable to energy systems. With the increase of different kinds of sensors and the integration of renewable energy sources, the grid data are high-dimensional, irregular and have external influences like weather, price influences, or regional load changes. Thus, static models and deterministic rules cannot provide operational reliability of modern energy systems.

When we check the literature we find that most of the development of commercial web based

energy management applications has been for visualization and reporting rather than adaptive intelligence. Traditional systems have in place dashboards which display variables such as power flow, voltage profiles and equipment states amongst other things, but none are equipped with automated analytics which can identify abnormal patterns in the network in real-time. Most of these interfaces are human interactive and not very scalable to handle large data. Their monitoring relies upon the use of historical baselines or the simple processing of statistical data, which usually results in slow reactions and low tolerance for faults. Lack of self-learning There is no mechanism for the systems themselves to learn of changing operational environments to produce the required changes; this is a fundamental barrier to evolution of these capabilities.

## 3.2 Proposed System

Based on the above-mentioned analyzing consequence, we propose Smart Grid Anomaly Detection System (SGADS) as a smart concept oriented Web-based solution for real-time fault detection and on-the-spot anomaly prediction domain. The method is based on deep learning, to model very general temporal relations and adaptive structural dependencies over multivariate gridded data. Normally, rather than using fixed thresholds and rules, it learns what it standard is by using historical data, and "raises an alarm" when things are out of the ordinary.

The literature of the SGADS is based on the Autoencoder with Attention Mechanism (AAM) which can learn compact latent representation during the reconstruction of data. We first train AE network on normal operating data to enable it to learn the intrinsic dependence among electrical parameters and environment in the normal operating situation. It then attempts to recompose this new found data according to what it already knows. The reconstruction is measured in terms of the difference between the input and the reconstruction output as a reconstruction error. Data points whose reconstruction errors are beyond a statistical limit were labeled abnormal. Moreover, having an attention mechanism it is also possible to investigate what are informative features and time steps both in terms of training and in decoding of the model. This choice of selection is favorable for interpretation and can be more sensitive (especially for time-dependent data and acting under temporal order) than an alternative bag-of-words selection.

SGADS makes its deep learning model available as a server in the cloud on a simple-to-install web framework using Python Flask. The architecture consists of four main layers, namely data acquisition, preprocessing, model inference and visualization layers. Pseudo real-time Smart meter, renewable, grid device sensor data streaming to the backend The preprocessing consist of scaling input, filling in the holes (with information from the neighboring time periods and timestamp to a common timestamp). The data after such pre-processing is then fed into an Autoencoder model to detect anomalies. Once an anomaly is detected, the model logs the timestamp of the anomaly, antecedents and confidence level. This is displayed in an interactive dashboard that gives an interface to operators in human-readable system performance. The platform is capable of not only real-time monitoring but also historical scanning so that anoma-

lies can be traced, explained and mapped back onto issues relating to physical phenomenon experienced in the grid.

### 3.2.1 Benefits

As we have argued in the Introduction, our systems provide advantages over traditional monitoring systems. Operates in real-time to analyze high frequency sensor outputs in order to trigger the anomaly seconds rather than hours or days after the explosive event It is due to the knowledge gained through deep learning and static rules that the system is able to adapt to changes in the grid dynamics without having to be recalibrated manually. The Autoencoder with Attention is an interpretable model in the sense that it is flexible. It graphically defines the nature of the event occurring (for example Some types of voltage disturbance, power factor deviation or temperature change are the main causes of generating abnormal conditions. This interpretive ability imparts to engineers and operators seeking knowledge of the physical reasons why what is occurring is so rather than simply a call to sound alarms. Second, the solution is web-based and utilized by distributed teams thus enabling cascaded monitoring of these assets as well as cross-functional coordination in a utility or energy company.

### 3.2.2 Solution Areas

The proposed solution overcomes major problems of existing energy systems. Regarding the fault detection, the SGADS can be used for detecting the overload in the system, the faulty transformer and the voltage instability before the occurrence of an outage. In the context of RE management, this method reveals irregular patterns between the production of solar energy and wind energy as these are influenced by environmental changes. It also helps in predictive maintenance by detecting changes in the operating conditions that precede a failure of the apparatus. Another solution domain is the dynamic pricing analysis: The system relates an anomaly in consumption to the electric price changes in order to identify energy waste. The analysis presented in the following can also be extended to apply to CPS other than electric networks like industrial production and smart water grids where detecting interhour anomalies in real time is equally essential.

## 3.3 Requirement Specifications

The SGADS requirements have been specified so that the deployed SGADS solution is accurate, scalable and robust, to the point of being used on real infrastructure. Each specification is the product of elaboration, validation, literature review; an isometric transitive closure, partly based on real data.

### 3.3.1 Purpose

The core issue of the Smart Grid Anomaly Detection System is to develop an intelligent analyzer that learns and adapts to the changing condition in the electrical network. Automated detection and diagnosis of anomalies during power operation is proposed, in order to allow the stability of the grid and to reduce downtime. As opposed to traditional tools that passively watch, or collect data, SGADS is active. The Web link turns it into an active decision-support system and thus not just a passive data warehouse. By using the deep learning and attention mechanisms, the proposed machine can offer prediction accuracy and interpretability for human experts to validate the results and modify the system threshold values whenever needed.

### 3.3.2 User Needs

The target users of this system are grid operators, engineers and managers who work on daily operation. Their requirement is for a system in which there never needs to be any (and certainly not much) manual adjustment. The user end should present the information so that it is easy to use, provide real-time exceptions and long term trends. The platform also appeals to data scientists or analysts who study and use storage of historical data formonitoring,remodeling the relationship,or optimizing parameters. Although the utilization from each user type is different, what is demanded for the system both share in common general requirements (e.g., reliability and transparency and speed). In a real-time grid environment where losses may occur due to every second of delay, latency and interpretability are as crucial as accuracy.

### 3.3.3 Assumptions and Dependencies

The success of the system relies on a number of factors related to the environment and infrastructure. We assume that the grid is equipped with a group of sensors that are extremely calibrated over time and successfully transmitting the same information. Whether it is the same or different network topology, it needs to link stable and safe data transfer. It is meanwhile assumed enough computational power during the training and real-time inference is able to train a deep hourly network with GPUs or high-end CPU.

#### Dataset

At the heart of this is the analytical capability that is utilized through the Smart Grid Real-Time Load Monitoring Dataset. It is time series reading trace of smart grid network at different time in a long time span. The database contains more than fifty thousand data records at 15 min time granularity and supports the real-time control application. Parameetres that are monitored are absolute potential difference, current, active power and reactive power. It also takes account of power that is available from solar and wind power sources, which are renewable, and "grid supply", which refers to the amount of power that is available through the main grid.

The information of overload, transformer failure is also stored. The environmental conditions (temperature, humidity, and electricity price) are also added to the dataset to get a richer description for contextual anomaly detection. The target of prediction and forecasting purpose is the kilowatt. As a result, it can be used in both unsupervised and supervised mode, and has a high coverage in terms of anomaly detection and energy saving.

### Hardware

The system has been developed and experimentally tested on a high end hardware platform based on an Intel Core i9 processor, 32 GB DDR4 RAM and a high end NVIDIA RTX 2080 GPU. These requirements enable efficient training of the deep-learning models, especially if large time-series data are used. Deployment STEM is surrounded by cloud infrastructure objects like using Amazon Web Service or Microsoft Azure objects which allow failing over and scaling. Even low-end hardware (e.g. multi-core CPU with 16 GB RAM) is capable of real-time inference with no GPU acceleration.

### Environment and Libraries

The machine learning and web integration work together with libraries and frameworks that are generally used in the development and implementation of the system in python. The backend of the web application is designed with Flask framework and it integrates the trained model with the visualizing layer. The autoencoder using Attention is built and trained using Tensorflow and Keras. Data analysis and pre-processing is done by Pandas and Numpy, explorative Data Analysis by Matplotlib and Seaborn. We used SqlAlchemy as well so we wanted to build on top of that something as lightweight as Sqlite as well. The ecosystem generates an open-source system that is fit for maintenance and can be extended by future research groups interested in developing such a system.

### Testing Requirements

Extensive testing is important to determine that SGADS is strong. Testing involves checking of the model itself, the latency and noise robustness. The unit test span input handlings, preprocessing, every way through the output formatting of every part in the data pipeline. Integration testing helps to validate prediction model serves to the dashboard properly. Performance tests serve to define the reaction time of the system with different loads, and stress tests - the system how many data streams it is ready to handle at one time. Artificial abnormalities and practical faults alarms is applied in the test of anomaly detection model for evaluation of quality.

### 3.3.4 Functional Requirements

The Smart Grid Anomaly Detection System should include secure user authentication, intuitive real time display mediated by a data acquisition process in the deep learning framework and dynamic dashboard visualization capability. Users need to be able to see current grid status, examine anomalies in the past and export analytical reports. Threshold adjustment, policy to have log or no log and alarm scheduling should be configurable in backend. Administrative users should have high level access to add end user accounts, change detection settings and view system health. All requirements are planned under the assumption of "state-in-the-art" operational grid components, enabling uninterrupted and autonomous operation in a live grid scenario.

### 3.3.5 Non-Functional Requirements

SGADS should be high-performance, scalable, and secure in addition to core functionality. It has to process the incoming data with a latency of less than two milliseconds so that the response is real-time. The accuracy of the anomaly detection should be more than ninety-five percents when it is normal. Secure The security was provided by encrypted communcation, hashed credential storage and input validation against injection attacks. The architecture of the system should be modular to accommodate future growth without significant re-engineering. Reliability and uptime are ensured through redundant database replication and container-based deployment. To the same degree, usability is our guideline; both technical and non-technical user can navigate through various process steps of the system with very low effort.

## 3.4 Use Cases

The use case analysis describes how various actors interact with the system and how the system responds in order to meet certain goals. The two dominant characters (i.e., actors) in SGADS are the user who is responsible for interacting with dashboards and understanding anomalies, and the administrator who deals with configuration details, users' information and overall system parameters. Every use case covers an individual mechanical functional scenario, in which the real-life application of the platform is represented.

### 3.4.1 Use Case Diagram

The use case diagram of SGADS is shown in Figure 3.1. The figure demonstrates that from the system, the users can access to inside and outside services such as: log in; monitor grid data entering and exiting; start up anomalies detecting task for various clusters or combinations of clusters; visualize results gotten; close to 2 minutes generate settings exporting reports.

Administrators, in addition to visitors' rights, have various rights including the right to change model parameters, create accounts and control anomaly.

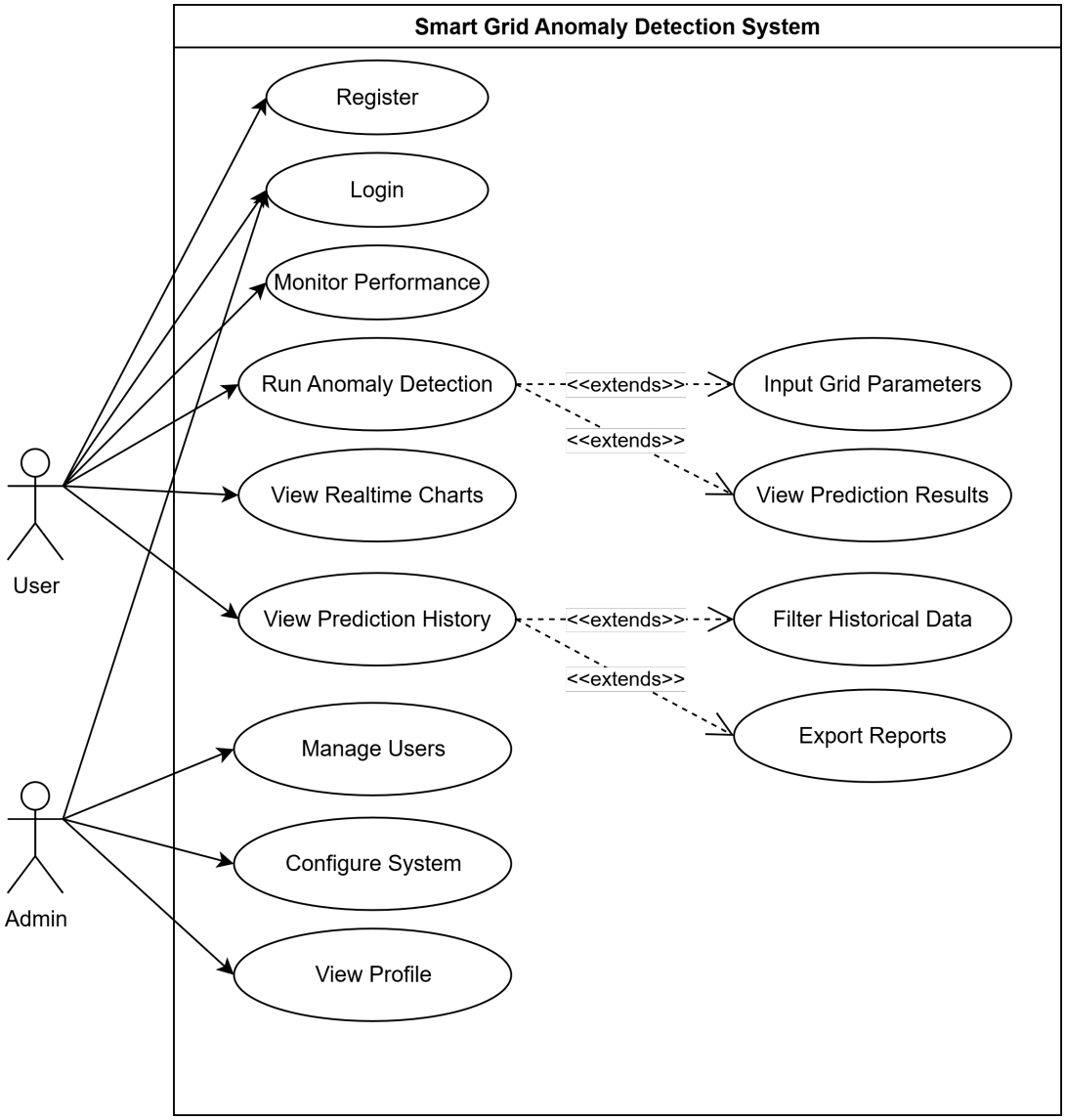


Figure 3.1: Use Case Diagram of Smart Grid Anomaly Detection System

### 3.4.2 Use Case Description

Usual use case is running anomaly detection process. The authenticated user logs into the web interface, and then chooses the data stream of interest. It acquires the recent readings, preprocesses them and passes them into a deep learning engine. Autoencoder is reconstructing and calculating the error with input. If the error is above an adaptive threshold, we cache such timestamp and set of attributes as anomalous. For both anomalies a visual alert is introduced in real time on the dashboard. The user can then review the context information with respect to the event and determine what, if any, appropriate corrective action is needed. All discordant results are retained for historical review.

Another use case is if admin are configuring model thresholds or retraining schedules. Some of them are to modify certain operational parameters, this can include altering alert sensitivity as well as managing access rights for a wide-range of users. Any change of configuration is trained and its recorded automatically into system for a true visibility. A second very helpful use case is spitting analysis reports. Users and administrators can generate reports on anomalies found over time windows in a summary format. The reports include statistical summaries, model confidence limits, and charts summarizing distributions of the anomalies. They're entirely exportable, in CSV and PDF so that you can use them for audits and compliance checks and whatnot.

### 3.4.3 Use Case Description Tables

The following detailed use case descriptions define the functional behavior of the Smart Grid Anomaly Detection System, from both a user and an administrator point of view. Each table specifies the actor, preconditions, postconditions and a sequence of steps which together define the scenario of a successfully performed use case. These tables provide the operational environment for the system as a whole.

Table 3.1: Use Case UC-01: User Authentication

| | |
|---|---|
| **Use Case** | UC-01 |
| **Use Case Name** | User Authentication and Access |
| **Primary Actor** | Registered User |
| **Pre-conditions** | The login-user is online, he has valid user credentials are in the database. |
| **Main Flow** | The user open the web application and provides credentials. the authentication module: the backend checks with credentials. If the validation is successful - a session and secure token will be issued. Dashboard interface is rendered allowing user to monitor the said features. |
| **Post-conditions** | The user is permitted into the authenticated environment with RBACP. |

Table 3.2: Use Case UC-02: Execute Anomaly Detection

| **Use Case ID** | UC-02 |
|---|---|
| **Use Case Name** | Anomaly Detection Processing |
| **Main Actor** | Authenticated User |
| **PreConditions** | The user is logged in and there are some live or historical grid data present in the repository. |
| **Primary Course** | The user clicks "Run Detection". The data is processed by the system, and is then passed to the Autoencoder with Attention model. The reconstruction error is calculated and the abnormalities that exceed the threshold value are stored. The dashboard is active and update in real time, displaying the timestamps of anomalies, confidence levels and the parameter it affected. |
| **Post-conditions** | Anomalies are detected, saved in the database and displayed on user dashboard. |

Table 3.3: Use Case UC-03: Visualization and Monitoring

| Use Case ID | UC-03 |
|---|---|
| Use Case Title | Visualization and Real-Time Monitoring |
| Primary Actor | Authenticated User |
| Preconditions | Anomaly detection engine is operational and data streaming is active |
| Main Flow | The user opens the visualization interface. The system sends anomaly detection outputs to the front end using asynchronous APIs. Graphs and charts get updated accordingly based on time evolution of voltage, load and anomaly scores. The user can zoom into particular slots or various variables and re-arrange views. |
| Postconditions | The user has visual data updates that can guide faster understanding and decision-making. |

Table 3.4: Use Case UC-04: Configuration Management

| | |
|---|---|
| **Use Case ID** | UC-04 |
| **Use Case Name** | Administration Configuration and Threshold Adjustment |
| **Primary Actor** | Administrator |
| **Preconditions** | The Administrator has logged in and has an elevated privileges. |
| **Basic Flow** | The admin opens the configuration module. The parameters of the system, such as threshold values for detection model re-training frequency, and alert sensitivity are shown. An administrator can then alter the specific setting he is interested. If no areas are specific, the backend will simply validate and send these updates to any connected clients without disrupting current sessions. |
| **Post-conditions** | The actual value of the configurations is saved to the database and directly processed in all modules. |

Table 3.5: Use Case UC-05: Generierung und Export von Berichten

| | |
|---|---|
| **Use Case ID** | UC-05 |
| **Use Case Name** | Generation and Export of Analytical Reports |
| **Primary Actor** | Authenticated User or Adminstrator |
| **Preconditions** | The system's storage layer contains historical anomaly data. |
| **Main Path/ Flow** | User selects option to generate report and inputs search by date range or category. It then queries those records which fit a specific time-frame and the results are processed, summarized into statistics and charts, to create downloadable reports in PDFs, CSVs or JSON format. |
| **Results** | The report is generated and can be accessed to retrieve or store it. |

Table 3.6: Use Case UC-06: User Management

| | |
|---|---|
| **Use Case ID** | UC-06 |
| **Title of Use Case** | User Management and Role Mapping |
| **Primary Actor** | The administrator |
| **Preconditions** | The administrator is logged in and user accounts are stored in the database. |
| | |
| **Main Success Scenario** | The admin opens the user management view, and checks a list of users registered and assigns or discards a role. Inactive users can be deactivated or their credentials can be reset by the administrator. Every change is audited for accountability. |
| **Postconditions** | The new list of users and their roles is securely and well organized stored. |

Table 3.7: UseCase UC-07: HealthMonitoring

| | |
|---|---|
| **Use Case ID** | UC-01 |
| **Name of the Use Case** | Monitoring System Health and Performance |
| **Primary Actor** | Administrator |
| **Preconditions** | The application and model serve are running. |
| **Main Flow** | System health module takes the administrator to the dashboard showing metrics like CPU usage, memory allocation, latency metrics and model inference rate. The admin monitors logs for all instances of abnormal flow of data or state in component. In case of abnormal operation, correct maintenance operation is realized. |
| **Post Conditions** | The administrator is provided with the real-time views of system performance and stability. |

These behavioural tables formalize the blue print of the system and act as intermediate between the conceptual design to the implementation. In this chapter, they define the user interactions in realisable operational terms, and maintain traceability to the functional and non-functional requirements specified in this chapter.

# Chapter 4

# System Architecture & Design

## 4.1 System Architecture

SADS What and Why The Smart Grid Anomaly Detection System (SADS) is a proposed multi-tier, service-oriented architecture to make it designed with modularity, scalability and maintainability in mind. Every layer has its own issues and is coupling with the other layers on well defined interfaces (another cornerstone of a good software architecture). This separation allows one to function and restrict testing without the parts being locked together and making it easier to work on them.

At the top we have our Presentation Layer which is the man/machine interface to like computing. It is the component that makes everything - everything you can see as frontends (dashboard component, analytic panel, authenticator page...) and admin part. Presentation Layer is responsive web-based so can be accessed using desktop and mobile browsers All of the times a user will do something, gridd parameter or historic data request, these are that your code lives and occur before Flask routing engine under the hood.

The coordination centre is in the Application Layer. It receives incoming http requests, looking and routing the responsibilities to its homologous controllers. The Flask router is coupled with several controllers like the Authentication Controller, the Prediction Controller and the Analytics Controller. Each controller is responsible for running operations like user authentication, invoking a model or running a data analytics query. The ability to stack this has enabled the use of user requests processing in parallel and the solution to non-blocking enabling transmission of real-time grid sensor data.

Business Logic Layer It is the logic used by a developer to perform various functions which can act on data available in the system. This layer supports the Anomaly Detection Engine, which is a combination of deep learning elements, feature engineering and anomaly detection monitoring frameworks. Consequently, we introduce a new Entity Assistant approach based on Auto-encoder with Attention layer, in which given the new stream of sensor data the model will try to learn such temporal dependencies and after that will determine occurrence of deviations

from the ideal working condition. This also covers the abnormal accumulation, which is the enumeration of individual continuous accumulation above the threshold to conceal a continuous event, such as transformer overloading and imbalances of the line.

All reads and writes can be handled in the Data Access Layer. It is layer that acts as an interface between code and the rest of the data storage (Model Loader, File Manager, Database Manager, etc.) This layer also ensures that the data is consistent and secure. There are importance of three service classes: Model loader that can read pre-trained deep learning model from repository, File manager to upload/down objects to/from secure storage and Database manager to read/write objects in database. The modules communicate with each other via transaction-safe queries which ensure atomicity, consistency, isolation, and durability (ACID) of data transformation.

The Infrastructure Layer is the more physical and infrastructure that supports both of the two above as well. This includes the SQLite database table used to store user profiles, logs of predictions and metadata about a configuration; the file system used to serialize models; and a ML model storage that has serialized learned model weights. This layer is manageable up to the cloud provider including 'spark-submit'. It means that distributed deployment, container scheduling and fault recovery are all available. The types of rack structure described For a design that means there should be clear separation between presentation, logic and data persistence code If the layered architecture is as it seems i.e: you have This is for maintainance over time and easy adoption to new technologies.

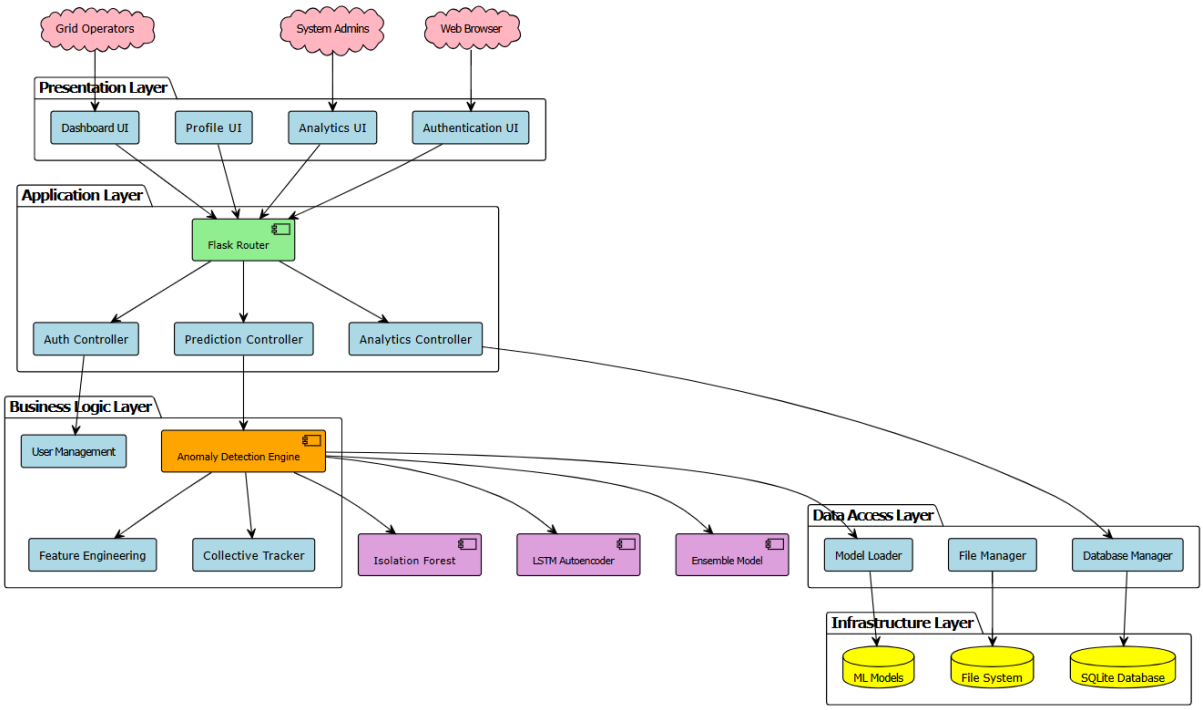


Figure 4.1: Layered System Architecture of the Smart Grid Anomaly Detection System

### 4.1.1 Context Diagram

The abscissas axis of the diagram shown describes the system operation scope and external interaction at a macro level. "It shows you how the different pieces interact to talk to this core anomaly detection engine. The external entities are the grid administrators, operators and the needed infrastructure like database or file system. The system operates on a web browser. They enter grid parameters or datasets taking real-time measurements on sensors uploaded. This input is then fed into the system and will provide in situ predictions on anomalies. Administrators use the system in an administrative manner, watching the one-shot analytics and changing model thresholds or seeing general trends of anomalies. Depend on external are, SQLite data base that contains predictions result, account of anomalies and users information as well as the file system that supports uploading profile images, datasets and serialized model files. The web browser also is the single point of access for all user interactions, that is, the architecture is immune to any targeted platform specific language. With a delineation of user and system responsibilities and role assignments, the context diagram assures that discipline will be maintained: The system will not stray from rigid data flow lines which organize users, processing modules, external entities.

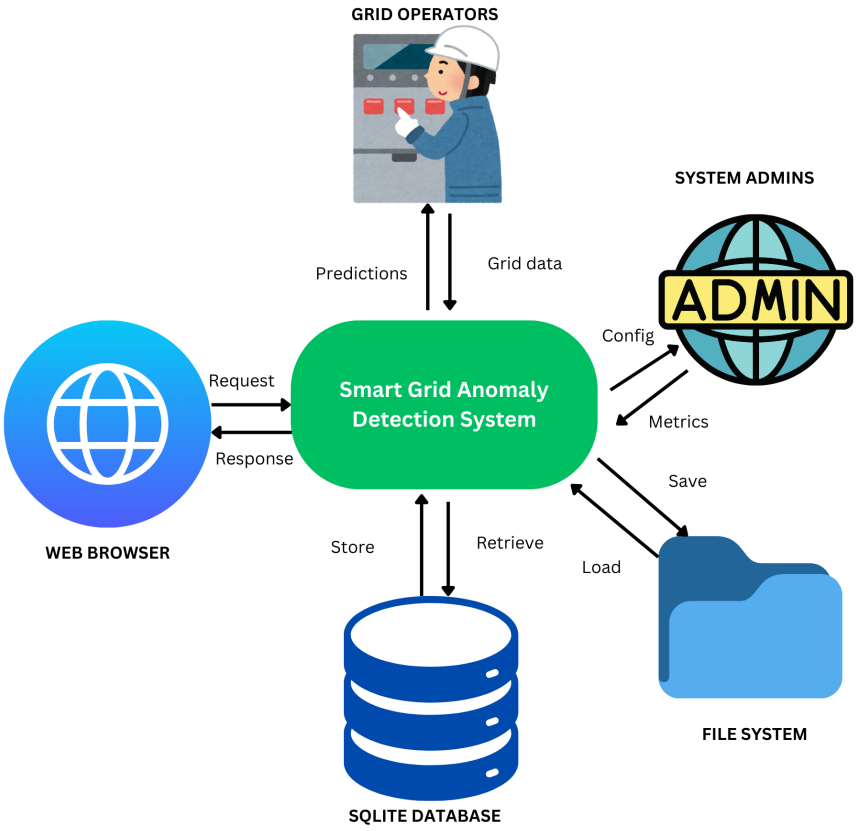


Figure 4.2: Context Diagram for the Smart Grid Anomaly Detection System.

## 4.2 High-Level Design

The higher-level design is organized around the modular decomposition and inter-modular communication. It is an account of how the building blocks collaborate to achieve the system's goals, but retain separation of concerns. The architecture is client-server based, in which the web interface is a client and a Flask application server is also the soul orchestrator. The interaction between the two is provided by HTTP request/response with JSON payloads, to support interoperability and ease of integration.

### 4.2.1 Component Diagram

The logical structure of the system is visualised as a component diagram in Figure 4.3. Every of these parts represents a software module that is based on some functionality and communicates with others explicitly using interfaces. All user inputs are processed by the Web Interface module and passed to Flask Application as RESTful APIs. The Flask Application determines the request type, and passes it to the appropriate submodule. The Anomaly Detection Engine, which is located in the business logic layer, runs the machine learning model pipeline. The Database Manager manages interaction with the SQLite database from which data is persisted and the File Manager that controls safe storage of model types and user uploaded resources. All these tools together form an integrated suite for anomaly detection, visualization and reporting.

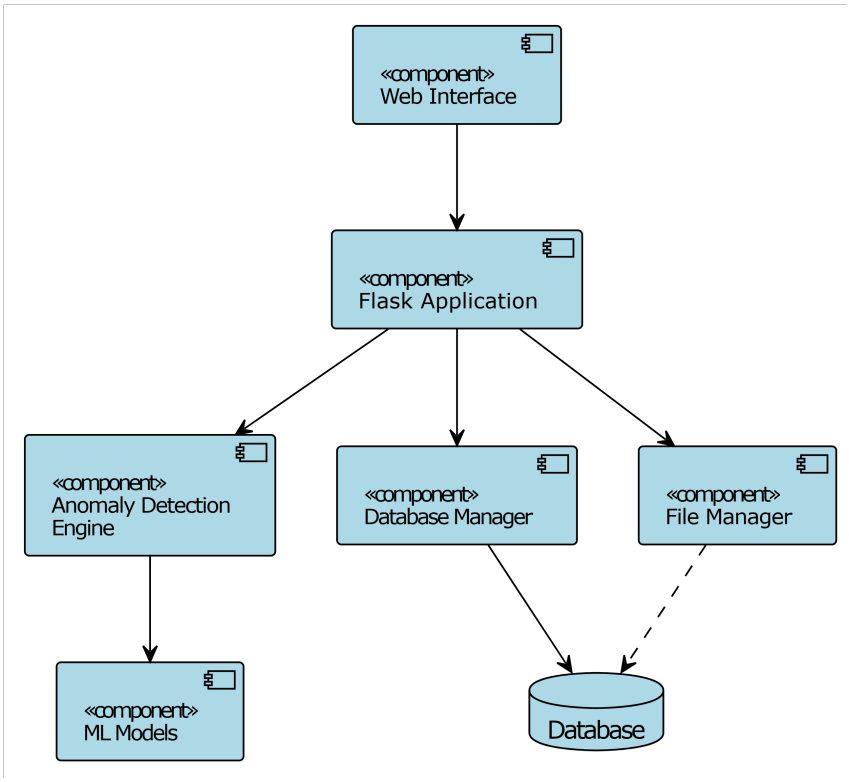


Figure 4.3: Smart Grid Anomaly Detection System Component Diagram

The Web Interface provides the view of the system, for both authenticated and administrative

users. It displays anomaly scores, graphs and analytic statistics in real-time. The Flask Application is the command center – it listens for incoming HTTP requests, delegates to the proper controller, and can coordinate application-level concerns. The component mainly responsible for feature extraction, normalization and inference is the Anomaly Detection Engine. It pre-loads the model and it makes a prediction on the new portion of data. Reconstruction errors are computed and the output is sent back to the flask layer for display. Prediction, anomaly timestamps as well user details are stored persistently by the Database Manager. Analytical queries may also be retrieved to form summaries and reports. The File Manager manages the profile picture, dataset and model checkpoint storage securely with versioning and integrity checks. The lightweight and trustful Persistence of SQLite Database ensures that the same can be bootstrapped with cloud PostgreSQL for horizontal scaling latter on. These associations are listed in Table 4.1.

Table 4.1: Component Descriptions

| **Component** | **Description** |
|---|---|
| Web Interface | As a user interface for user interaction, real-time dashboard and data visualisation via responsive web pages. |
| Flask Application | The primary switch handling routes, user sessions, and interaction between UI and back-end services. |
| Anomaly Detection Engine | Runs deep learning models and performs preprocessing, feature extraction, and collaborative anomaly analysis. |
| Database Manager | Responsible for the orderly storage and retrieval of predictions, user accounts and analytics logs. |
| File Manager | Administrates file operations such as uploads, downloads, and securely managing datasets and model artifacts. |
| Machine Learning Models | Pre-trained Autoencoder and Attention with auxiliary statistical models used during inference. |
| SQLite Database | Persistent store used for structured information like predictions, config data, and user meta-data. |

## 4.3 Low-Level Design

The fine-grained design addresses its working. It defines internal processes, processing of data and exchange of messages among elements. This part also serves to point out where the machine-learning processing is integrated with the rest of a web application.

The procedure of anomaly detection starts when a user uploads new grid data. The Flask controller handles this request, it checks whether the user has permission to use the service and it sends the data into the pre-processing module. Data normalization and missing values

interpolation are performed at this step. Pre-processed data is sent to the Autoencoder which reconstructs inputs for calculating the reconstruction error. The error is thresholded against a dynamic threshold. When the threshold is overpassed, we label the data point as an anomaly and send it to the analytics module for visualization and save. Each of these steps obeys a specific process described by activity and sequence diagrams.

### 4.3.1 Activity Diagrams

6.3 Activity diagrams The activity diagram is a dynamic model that shows the dynamic flow within the system. They illustrate control and data movement between stages during different operations.

The first activity diagram is the collective anomaly tracking process depicted in Figure 4.4. Each time the Autoencoder identifies a sequence of consecutive anomalies for some specific user or grid node, an internal counter is incremented by the system. An event was identified as a collective anomaly when the count is over 5 consecutive anomalies. This feature allows long-term fault detection, where short-term spikes are suppressed but persistent anomalies produce alerts.

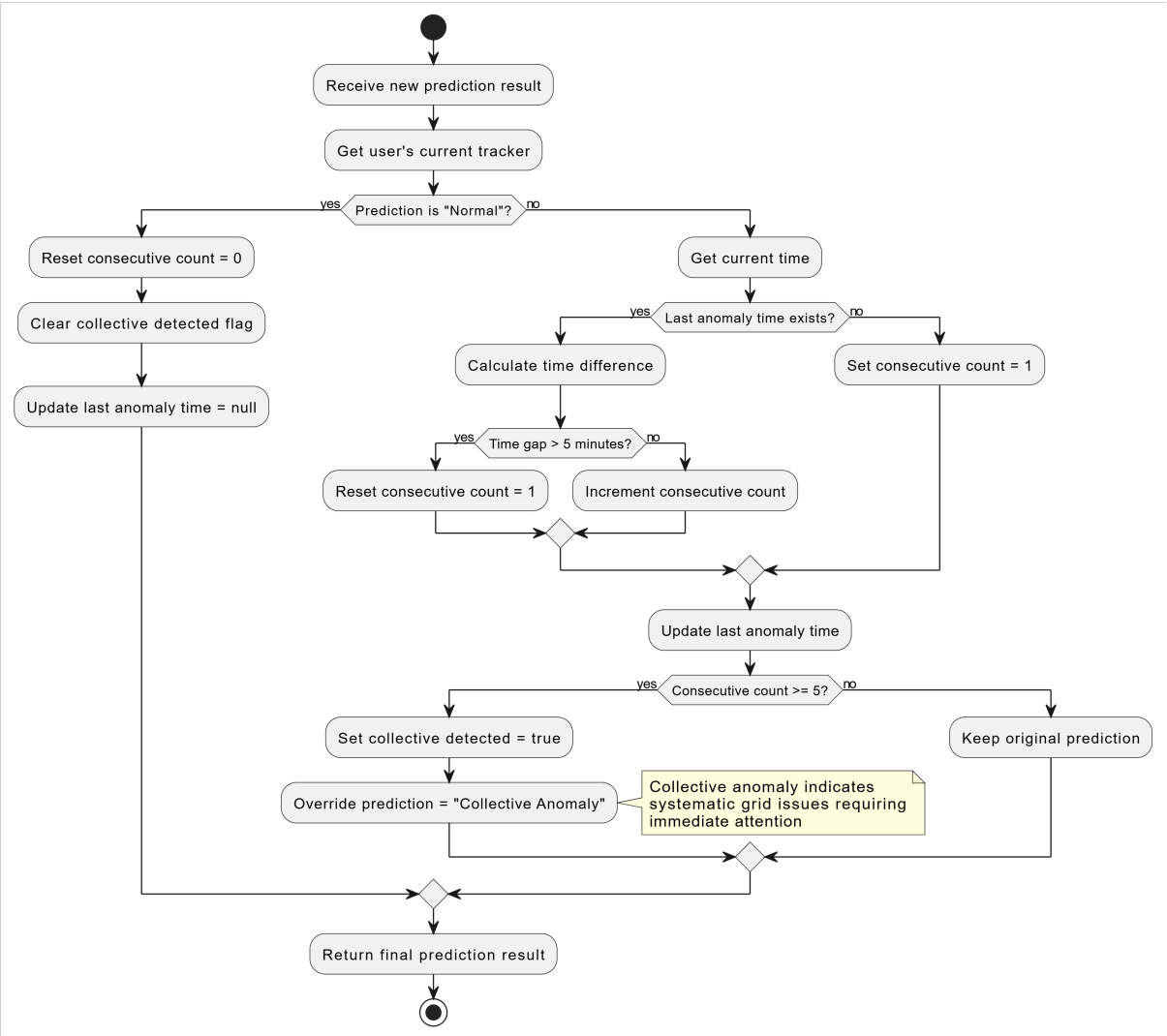


Figure 4.4: Activity Diagram: Collective Anomaly Tracking

The process of activities in the analytics dashboard, from user authentication to obtaining data and visualization is summarized in Figure 4.5. Upon successful login, the flask application verifies whether the provided credentials are correct and retrieves history of prediction. The data is then rearranged to an object which can be interpreted as JSON and visualized with Chart. js. The dashboard auto-refreshes asynchronously to provide near real-time responsiveness.

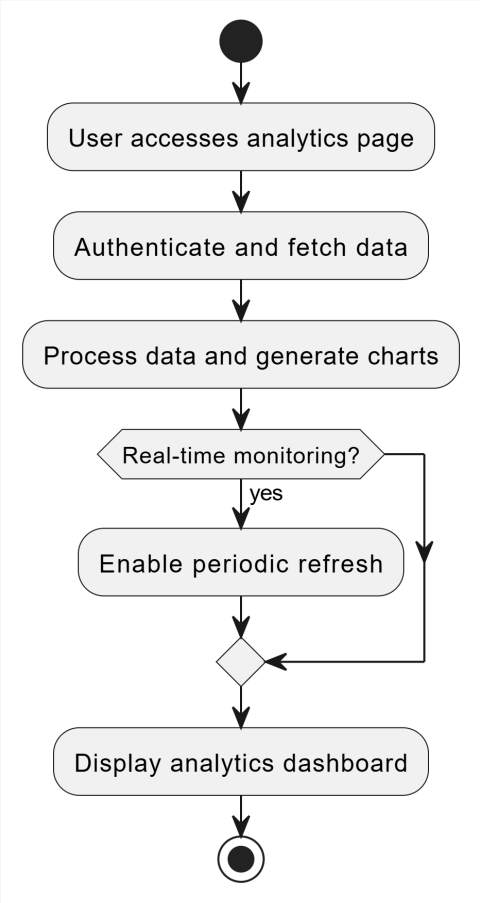


Figure 4.5: Activity Diagram: Analytics Dashboard Workflow

The third figure FIGURE. 4.6, illustrates the abnormal activity detection from input to output. The procedure starts with the user entry of voltage, current and load values through the dashboard. These values are checked for presence and validity, then transferred to the Flask webapp. The Autoencoder model is called for inference by the backend and the resulting scores are returned to the user interface for display. This process is designed to make predictions reliable, timely and traceable.

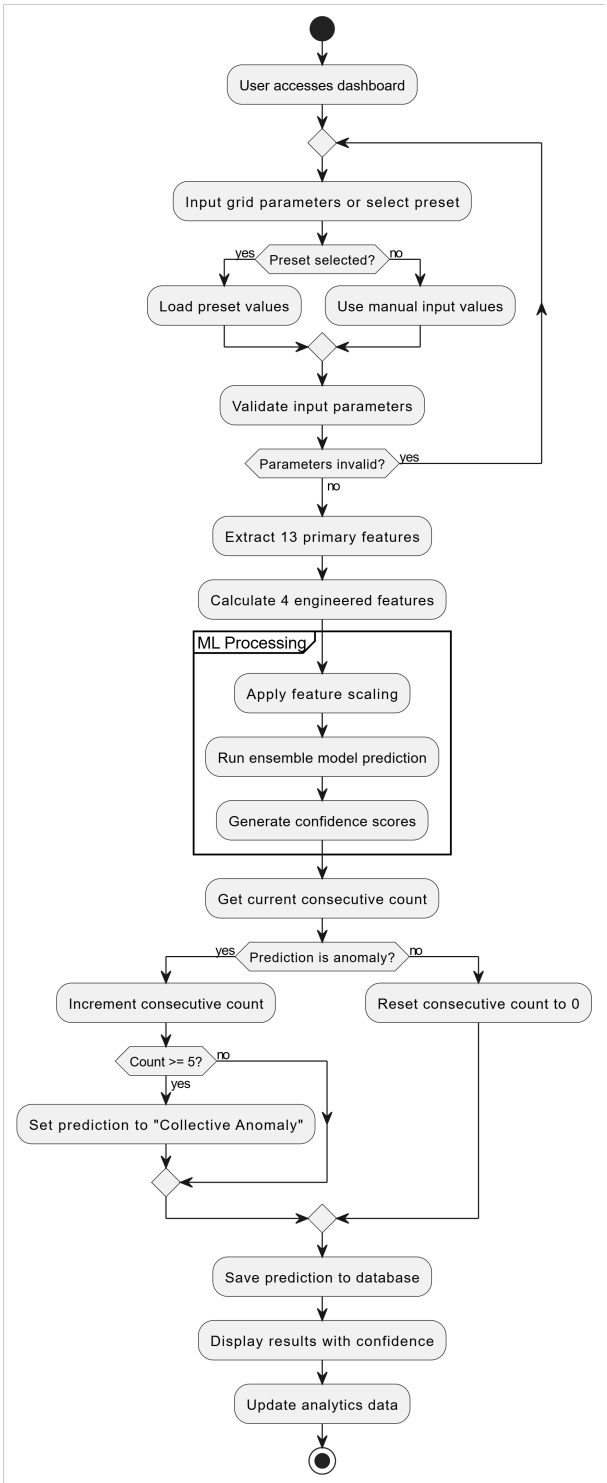


Figure 4.6: Activity Diagram: Anomaly Detection Process

## 4.4 System Sequence Diagrams

Sequence diagrams represent the temporal ordering of how messages are passed across system components. These are also useful for timing constraints and event-driven behavior. The first sequence with a diagram (see Fig. 4.7) shows the updating a profile. Client On-the client, when

a user updates their profile/profile picture, there is an update request sent to the Flask back end. The request is accepted by the system, a new image is stored on disk and a change is made to the database. Upon completion of a transaction, a message indicates on the user interface that is authorizing to publishall operations both.

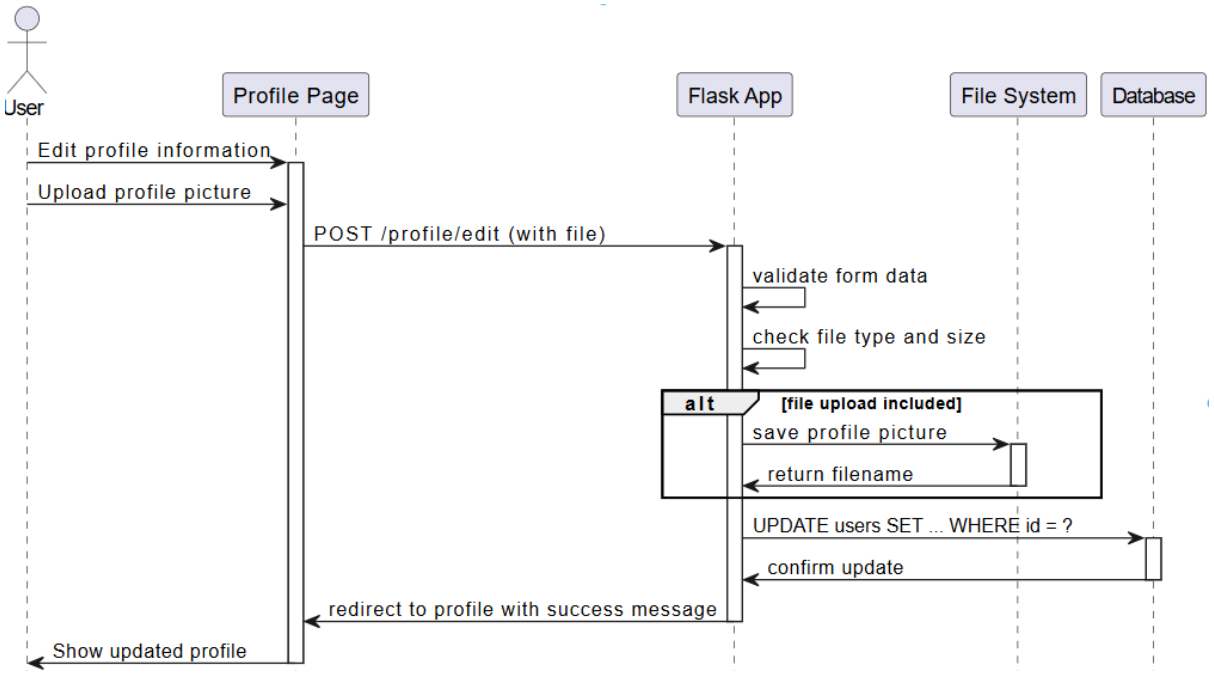


Figure 4.7: Sequence Diagram: Profile Update Process

The sequence-of-sequences illustrated in Figure 4.8 is that of retrieving the analytics dashboard, as from this sequence of playback flows and messages we see how user input turns into analytical data. It all begins when a user opens the dashboard. The Flask server checks a request by an API key and queries the database for records of predictions; afterward, it replies as JSON. The data is then returned to the client-side, and displayed as interactive charts. This pipeline ensures the backend analytics and frontend visualization are always in lock step.

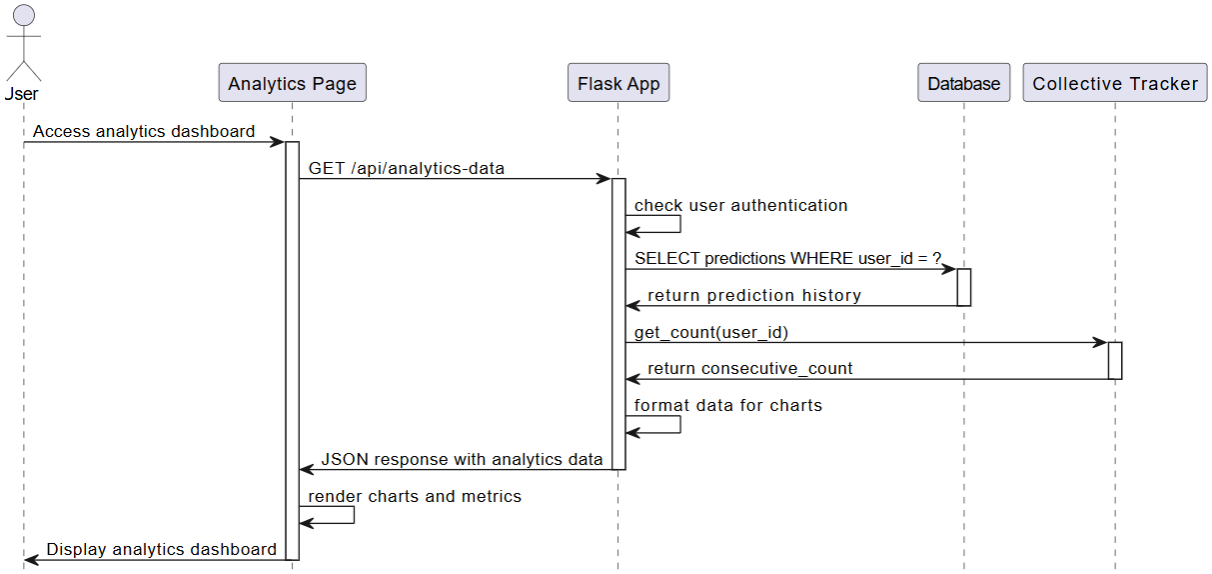


Figure 4.8: Sequence Diagram: Analytics Dashboard Retrieve Data

User Authentication46, for example as illustrated by the sequence in Figure 4.9, is the base of a secure access control. The user logs in using the login page, s/he provides credentials that are sent over HTTPS to the Flask authentication controller. The passwords are hashed and then compared the db with stored hashes. Successful authentication would generate a secure token, store it and return to the client. Later requests keep using this token for stickiness.

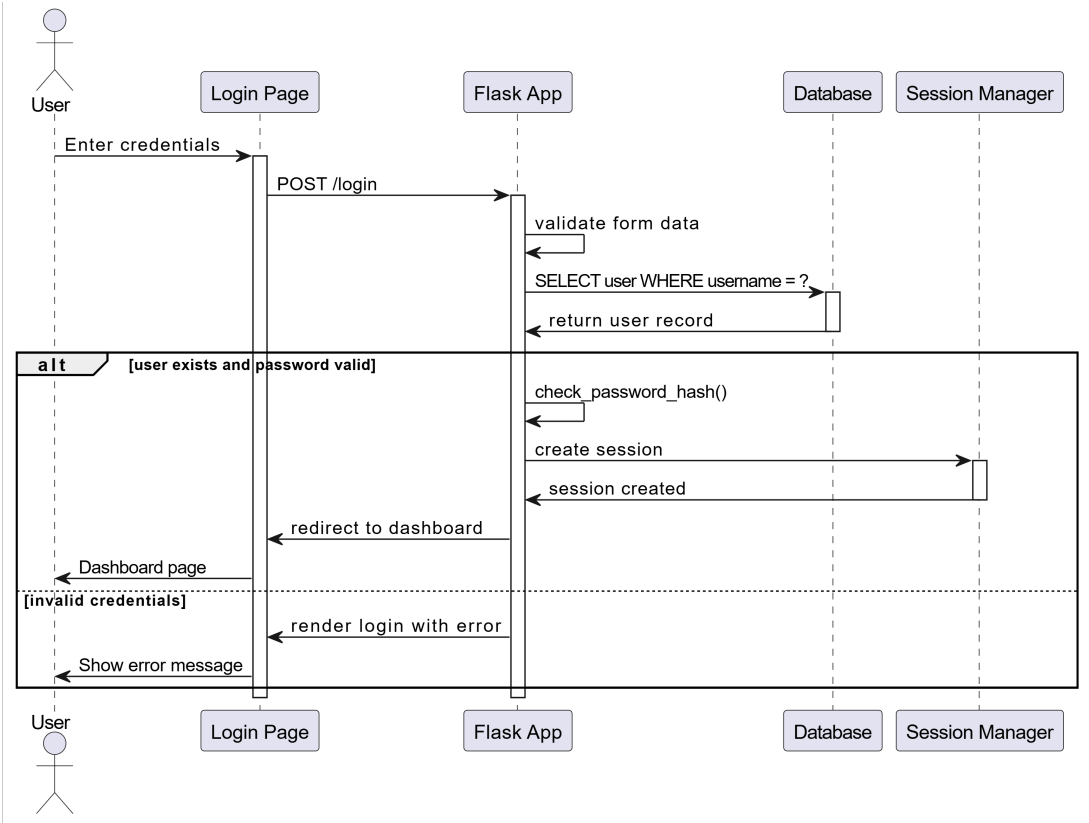


Figure 4.9: Sequence Diagram: User Authentication

Figure 4.10 shows the sequence diagram for the overall process of detecting anomalies, which connects all steps in the processing of data to final anomaly generation. The user enters the parameters in the interface and preprocessing is started from the back end. The processed data is then submitted to the Anomaly Detection Model which makes use of the trained Autoencoder to make inferences. Then the reconstruction error and anomaly score are sent to analytics module, saved into database and presented on user's dashboard. This organized communication provides end-toend traceability and accountability for all predictions.

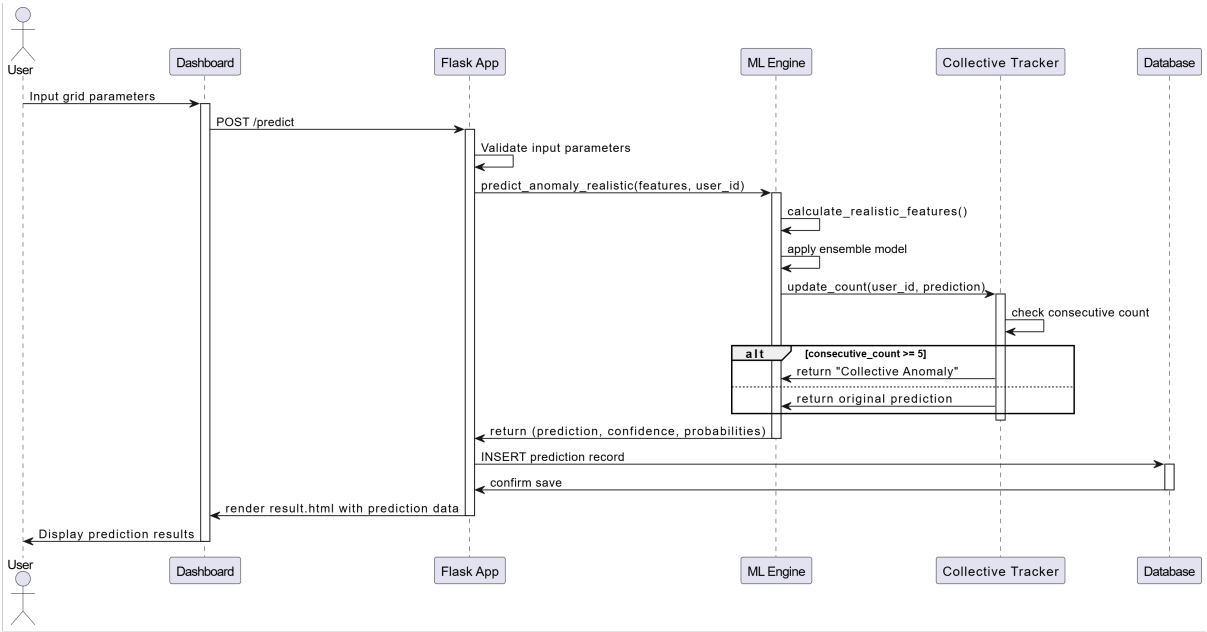


Figure 4.10: Sequence Diagram: Anomaly Detection

# Summary

This chapter introduced in detail the design and framework of Smart Grid Anomaly Detection System. The system architecture was deeply analysised, including separating presentation layer from application layer, application from business logic and so on. Co-actions were introduced to explicate system boundary and external relations. The sections on high-level and low-level design showed how data is passed through components, how activity diagrams represent operational workflows, and how sequence diagrams illustrate role based communications in real time between modules. This unified approach provides for scalable, modular and transparent solutions and forms a solid base of the development and evaluation in the following section.

# Chapter 5

# Implementation

## 5.1 Overview

In this chapter, the implementation of the complete Smart Grid Anomaly Detection System, frontend and backend implementation, and machine learning, are all discussed. We created this system as a web-app using the flask microframework, and locally deployed it in order to dynamically try and evaluate our design in an iterative way. The implementation process was developed in order to be modularity and layers-based, meaning that each subsystem involves from preprocess raw gridded data, visualize real-time anomalies, is indepentent and maintnable, extensible and testable.

The whole end-to-end workflow is ultra score computing, data import (ingestions), data pre-processing, feature engineering, anomaly detection and classification. At the backend, our applications are implemented in Python (3.10) and using Flask, Pandas and NumPy with Scikit-learn for Data Manipulation & Inference latterly, TensorFlow for our Deep Learning layer offers (LSTM Autoencoder etc.). This is to ensure that user data and prediction history are stored in a SQLite3 database, and this also means that you can be relatively certain of it working as expected.

We choose flask instead of heavy power hogging frameworks like django cause we wanted to have routing tree in our hands so that lag between server and client is minimized. It is little architected and slots cleanly into Python's scientific computing stack and allows the same runtime to perform web services to model the inference. In addition, Flask also has no problems working with `joblib` and `keras` when serializing models and for loading the trained artifacts at the runtime. The full-blown system can be run on a local development server available at `localhost:5000` so it is possible to test and debug the code separately before moving to production servers.

## 5.2 Development Environment

We conducted all implementation and testing on a personal workstation with Windows11, using Python3.10. The system consisted of an Inteli7 (9th generation) processor, 32GB DDR4 RAM and NVIDIA RTX-2070 video card. While the majority of computations in the end-to-end system are performant on CPU, during the training for LSTM Autoencoder phase, GPU acceleration was leveraged to optimize matrix operations and minimize time to converge.

We used `venv` to create a virtual environment for depedency isolation. The essential libraries I installed were Flask, pandas, scikit-learn, tensorflow, matplotlib and chart. js for visualization integration. I recorded each in a `requirements. txt` file for reproducibility and easier environment duplication in different environments.

Flask and Http Logging are made possible through Python's `logging` module Every HTTP request, model inference is logged into a rotating log file as well as database transaction. It has the benefit of being able to trace for debugging and post-audit analysis. Version tracking was set up with GitHub so that every step of model training, parameter tuning and UI improvement that we took is traceable.

## 5.3 Data Preparation and Feature Engineering

The Smart Grid Real-Time Load Monitoring Data used in this document comprises of more than 50k data points which are timetamped as every 15 minutes. At each node of the power grid, the electrical and environmental parameters constituting the state of the operation are logged. Some of the features are voltage, current, active and reactive power, power factor; solar wind generation alongwith grid supply; ambient temperaturehubidtyehlectricity price. In addition, renewable energy sources such as wind and solar add to fluctuations in the supply and demand for electricity giving AD a hard but feasible application.

### 5.3.1 Data Cleaning

Before we trained our models, we investigated the raw dataset for inconsistencies, outliers and missing readings. To address the missing value problem, we performed linear interpolation in the temporal dimension that would ensure intermittent free interpolation in time between column adjacent time frames. Outliers where the aborted antenna or spurious transmission noise occurred were detected with the help of interquartile range based thresholding, and smoothed using rolling median (window size of 3). This has the advantage of not corrupting the statistical nature of the training data, and it would not allow extreme spikes to popularize its distribution. Time stamps was one hot encoded and converted to chronological index for chronological order of time in LSTM Autoencoder. Furthermore, each recording was also checked for unit coherence to ensure that all voltage, current and power were measured using

standard SI units.

### 5.3.2 Feature Engineering

To better capture the physical traits of interest, additional engineered features are built. Furthermore, four other artificial features based on the thirteen 'basic' ones were added for system representation: RenewableTotal (sum of the sun and wind power), RenewableRatio (RenewableTotal over total consumed electrical energy), Power–VoltageRatio (the ratio of active power and voltage) and Efficiency (the product topological differences with l-index > = 2 utilities). These derived attributes represent domain knowledge about energy conversion efficiency and the proportion of the renewables.

Listing 5.1: Feature engineering before training.

```
def engineer_features(df):
df = df.copy()
df["renew_total_kw"] = df["solar_kw"] + df["wind_kw"]
df["renew_ratio"] = (df["renew_total_kw"] /
(df["power_kw"] + 1e-6)). clip(0, 5)
df["p_v_ratio"] = (df["power_kw"] /
(df["voltage"] + 1e-6)). clip(0, 10)
df["efficiency"] = (df["power_kw"] /
((df["voltage"] * df["current"]) + 1e-6)). clip(0, 1.5)
return df
```

Following engineering, this dataset is standard-scaling with the `StandardScaler` implemented in scikit-learn. Each feature is decorrelated so that it has zero mean and unit variance, which helps models from being biased towards features with larger numeric range. The fitted scaler over the training data is saved with `joblib` and then reimplemented during inference to keep things consistent.

## 5.4 Backend Implementation

Backend Responsible for handling user request to that serve the and output and forward this data further to machine learning models and then receive the predictions to give it back to frontend. It's built on Flask and has multiple registration modules: auth, anomaly detection, history tracking and analytics.

When Flask sends a request into the /predict endpoint it reads the JSON payload in, validate input features, does some processing like we did during training and sends pre-processed data back to model inference. Everything that happens between the time it gets that request, and when it sends a response, all occur in sequence within one major round of HTTP; that leaves the server stateless and breathing easy.

Listing 5.2: A simplified prediction route

```
@app. route("/predict", methods=["POST"])
def predict():
payload = request. get_json(force=True)
df = pd. DataFrame([payload["features"]])
df = engineer_features(df)
X = df[PRIMARY + EXTRA_FEATURES]
Xs = scaler. transform(X.values)
proba = gb_model. predict_proba(Xs)[0]
label = gb_model. classes_[np. argmax(proba)]
return jsonify({
"result": str(label),
"confidence": float(np. max(proba))
})
```

For each prediction the time, input features and predicted label/ model confidence are saved in the SQLite database. The database also stores group anomaly counts and user authentication information. Any database transaction members are committed atomically ensuring that, in the event of an unexpected failure, no incomplete data is stored.

### 5.4.1 Collective Anomaly Tracker

The system distinguishes between individual anomalies and persistent collective deviant behavior. A collective anomaly is detected when several subsequent outliers occur, one after clear another. To monitor this, a rolling anomaly count is maintained for each user session in memory. A group anomaly is recorded if the count exceeds five.

Listing 5.3: Collective anomaly tracking mechanism.

```
class CollectiveAnomalyTracker:
def init(self, window=5):
self.window = window
self.store = {}
def update(self, user_id, is_anom):
rec = self. store. get(user_id, {"count": 0})
if is_anom:
rec["count"] += 1
else:
rec["count"] = 0
self. store[user_id] = rec
return rec["count"]
```

This method reduces the overhead from the bandwidth when updating to a database, as an update happens only when a counter reaches its threshold. It can online monitor the health condition of system and the total frequency of anomaly effectively.

## 5.5 Machine Learning Pipeline

### 5.5.1 Isolation Forest Training

The base model of the ensemble is the Isolation Forest, which is an unsupervised algorithm that isolates anomalies by separating portions of data to minimize the information to isolate anomalies. It does not depend on distance and density assumptions, thus is appropriate for high-D data. To train the model, 200 trees were built with a contamination rate of four percent to estimate the fraction of anomalies in the dataset.

Listing 5.4: Isolation Forest model fitting.

```
iso = IsolationForest(n_estimators=200,
contamination=0.04,
random_state=42)
iso.fit(X_scaled)
iso_score_train = -iso. score_samples(X_scaled)
joblib. dump(iso, "artifacts/iso_forest. joblib")
```

The negative score output of the Isolation Forest used as an auxiliary feature in the meta-set for the ensemble classifier.

### 5.5.2 LSTM Autoencoder Training

The LSTM Autoencoder is able to model temporal dependencies of the grid data. Six hour segments (24 time-steps) are taken with sliding windows with overlap. The network is composed of an encoder LSTM and a decoder: The former compresses the input into a latent representation, whilst the latter reconstructs the sequence. The model is learned to minimise the mean squared reconstruction error, and thus it captures how peculiar a considered sequence w.r.t. normal behaviour indirectly.

Listing 5.5: LSTM Autoencoder architecture.

```
ae = Sequential([
LSTM(32, input_shape=(SEQ_LEN, INPUT_D)),
RepeatVector(SEQ_LEN),
LSTM(32, return_sequences=True),
TimeDistributed(Dense(INPUT_D))
])
```

```
ae. compile(optimizer="adam", loss="mse")
```

Eighty percent of the samples were used for training with early stopping using validation loss to prevent overfitting. A serialised Keras `. h5` file for later inference. At test time, the reconstruction error of each of the sequences is also a meta-feature to Gradient Boosting model.

### 5.5.3 Gradient Boosting Ensemble

In the last stage, Gradient Boosting algorithm is applied, which aggregates several weak decision trees into a strong predictor. Each iteration trains a new tree on the residuals of the previous one, progressively reducing error. The ensemble has 200estimators, a learning rate of0.1, and maximum depth of 3. It produces class probabilities for four classes: Normal, Point, Contextual and Collective anomalous samples.

Listing 5.6: Gradient Boosting classifier training.

```
gb = GradientBoostingClassifier(n_estimators=200,
learning_rate=0.1,
max_depth=3,
random_state=42)
gb.fit(meta, labels)
joblib. dump(gb, "artifacts/gb_ensemble. joblib")
```

### 5.5.4 Unified Inference

At testing time, three trained models act collectively. The Flask service loads every models at start, and then we load each model features to calculate as: 1) engineered features + scaling them 2) Isolation Forest score 3) Autoencoder reconstructed error For all the sub-models, we create a combined vector that goes to the Gradient Boosting classifier.

Listing 5.7: Unified prediction pipeline.

```
def predict_vector(x_row, seq_buffer):
df = engineer_features(pd. DataFrame([x_row]))
Xs = scaler. transform(df[ALL_FEATURES]. values)
iso_score = -iso. score_samples(Xs)
seq_buffer = np. vstack([seq_buffer, Xs])[-SEQ_LEN:]
ae_err = float(((ae. predict(seq_buffer[None,...]) -
seq_buffer[None,...])**2). mean())
meta = np. concatenate([Xs. ravel(), iso_score, [ae_err]])[None, :]
proba = gb. predict_proba(meta)[0]
return {"label": str(gb. classes_[np. argmax(proba)]),
"confidence": float(np. max(proba))}
```

This cascaded inference architecture achieves robustness through fusion of several independent detection modalities.

## 5.6 Database and Persistence

The persistent data store that it's built on is SQLite. Its schema features tables such as `users`, `predictions` and `analytics`. Table *Predictions* contains for every request its timestamp, parameters, predicted class and confidence. Using SQLAlchemy ORM makes it easy to use with the Flask models and a database table without having to worry about low level SQL query.

Artifacts like the trained models and scaler are saved in `artifacts/` with time-stamped filenames to track versions. You can retrain a model and the new version is uploaded automatically, with older versions retained in case you need to roll back. This allows for safe updates without disturbing the ongoing service.

## 5.7 Frontend Implementation

User Interface The GUI of the Smart Grid Anomaly Detection System serves as the interactive interface between the user and analytical engine. It is designed to be clear, responsive and accessible so that any user - technical or not - can make sense of the system's outputs. The interface is implemented with HTML5 for structure, CSS3 for style and Bootstrap 5 for responsive layout. JavaScript and Chart. js allow for asynchronous data updates and visualisations. Every page on the app is templated using Jinja2, to render backend data dynamically and establish a stylistic coherence throughout the website.

The designing was done in a layered manner with the Dashboard view acting as the operational input hub, Result page displaying direct prediction outcome, History page filled and storing user's interactions and predictions while Analytics being used for high-level insights and anomalies statistics. An additional Administrator panel is provided to administer system activity, adjust model thresholds and export logs.

### 5.7.1 Signup Page

The system starts with the user registration page for new operators or administrators to sign up an account which will be used to view the anomaly detection. This form gathers required info to create user: name, department, email address, password and optional image. Data on the client and server are validated for data integrity.

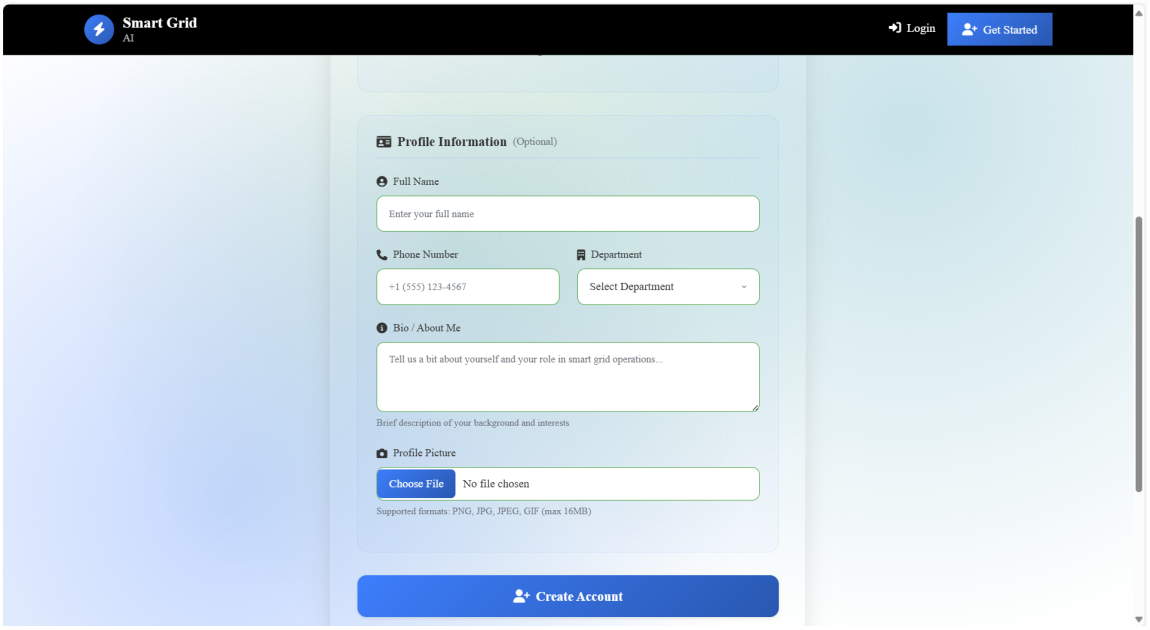


Figure 5.1: User information form and validation hints on signup page.

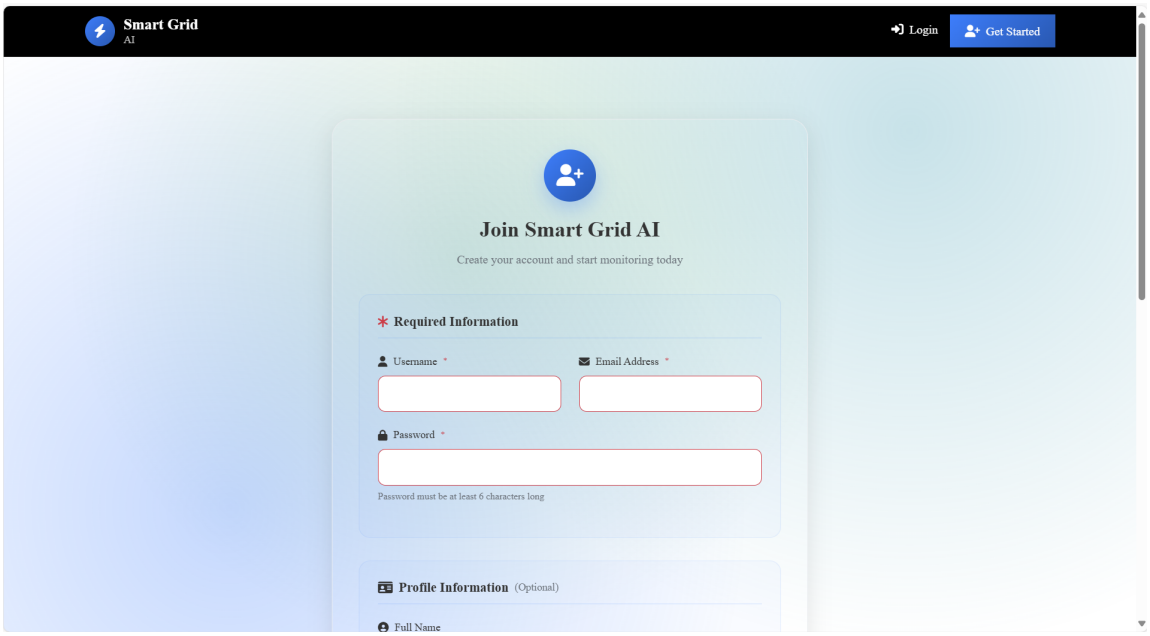


Figure 5.2: Signup page with more fields like department and profiles picture upload feature.

When signing up, the password is hashed and saved in a SQLite database. The application using some security measures implemented in Flask to avoid SQL injection and cross-site scripting.

### 5.7.2 Login Page

Upon successful registration, users authenticate using the login page that checks the credentials with those saved in the database. The application is authenticated in session, and login states

persist until you log out. Invalid logins will generate descriptive alerts so you don't need to worry about unauthorized attempts.

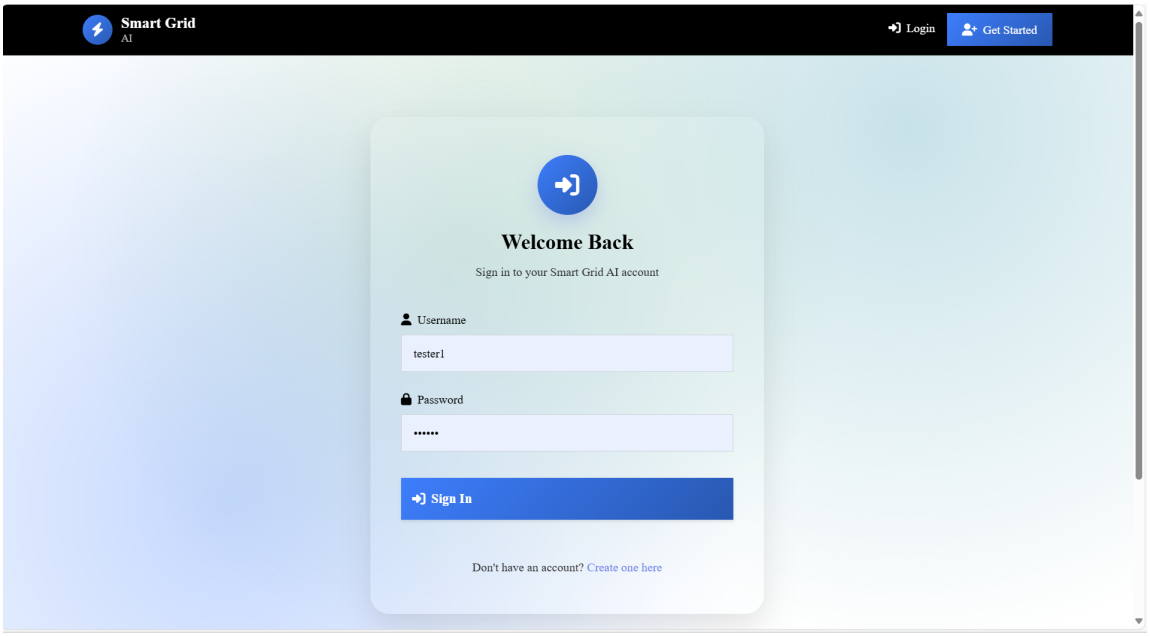


Figure 5.3: The log-in page for user authentication and secure access to the system.

Your administrators are directed to the monitoring console, and your users go straight to their main dashboard when you log in. Authentication with a Flask session system, using signed cookies to wrap the user's Session.

### 5.7.3 Dashboard Interface

The Dashboard is the user's main point of entry. It consists of thirteen moving sliders and text entry boxes which can be adjusted to model different grid parameters such as voltage, current, power consumption and renewable energy. The controls can be manipulated by the user to model or enter real-time information. Derived Quantities like the (Renewable) Ratio and Efficiency are calculated on-the-fly and shown below the inputs. This powerful feedback allows users to learn about how grid performance changes as parameters vary.

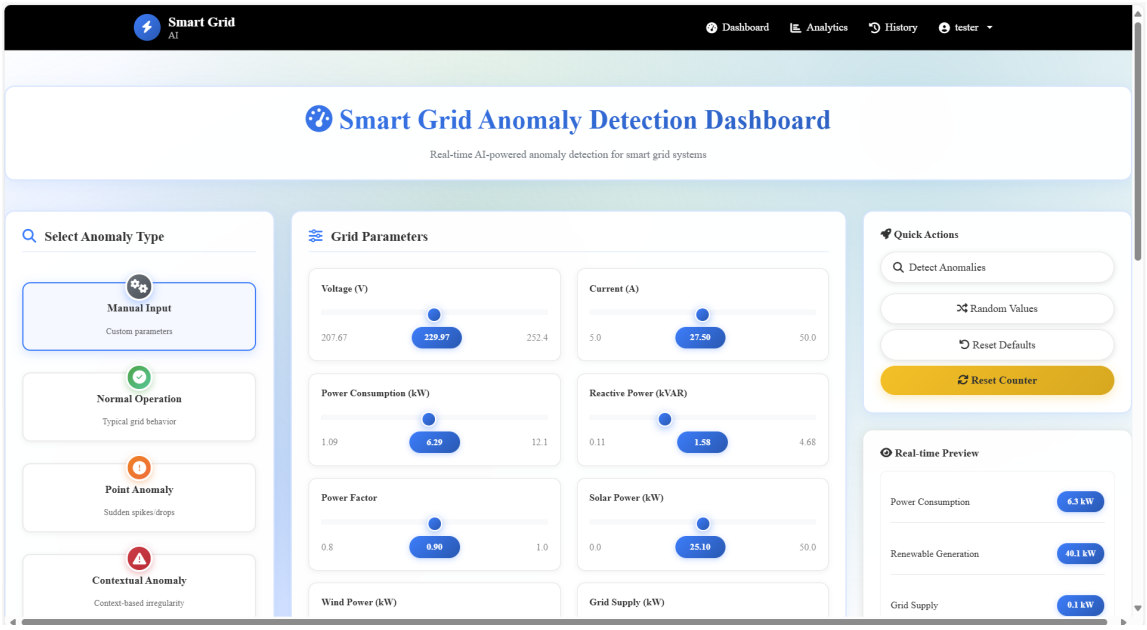


Figure 5.4: Dashboard Interface: displayed parameter sliders, real-time metrics and system-status indicators.

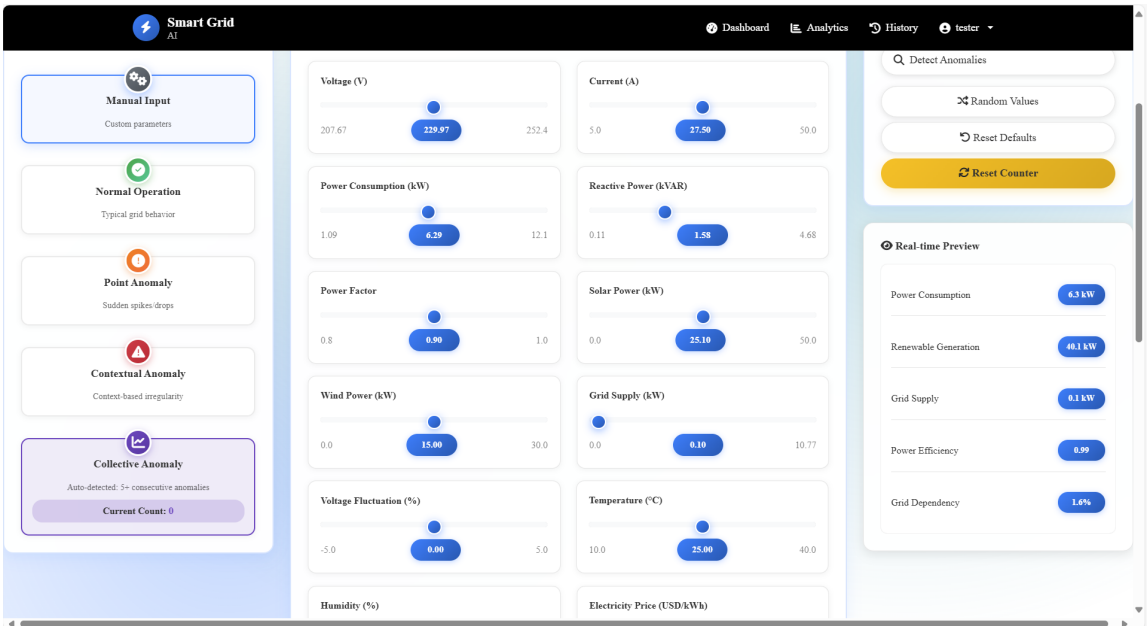


Figure 5.5: Extended Dashboard with extra Sliders for parameters and adjusted efficiency calculations.

Quick summary panel: Below the Dashboard, there is a small summary section of the quick analysis chart which presents the most recent five prediction anomalies and their types. This gives an immediate operational context without the need to scroll down into the history.

### 5.7.4 Prediction Result Display

When the form is submitted by the user, a POST request is made to the Flask endpoint from the frontend using AJAX. The backend sends a JSON object with predicted type of the anomaly, probability distribution and confidence score. Responses are immediately displayed using asynchronous JavaScript calls, so it is a projective real-time system.

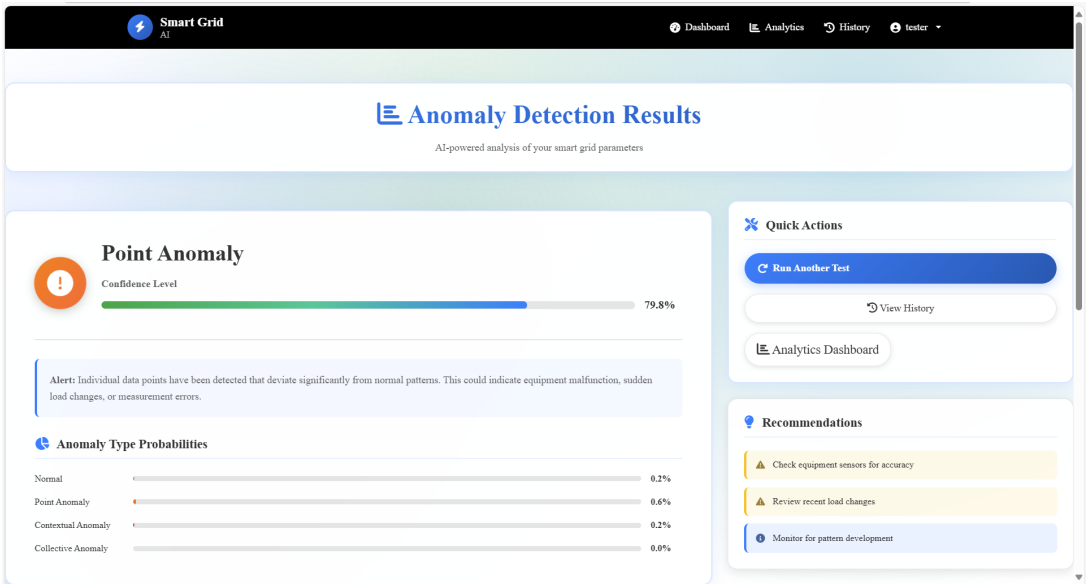


Figure 5.6: Prediction result panel that show the anomaly class, confidence score and probability visualization.

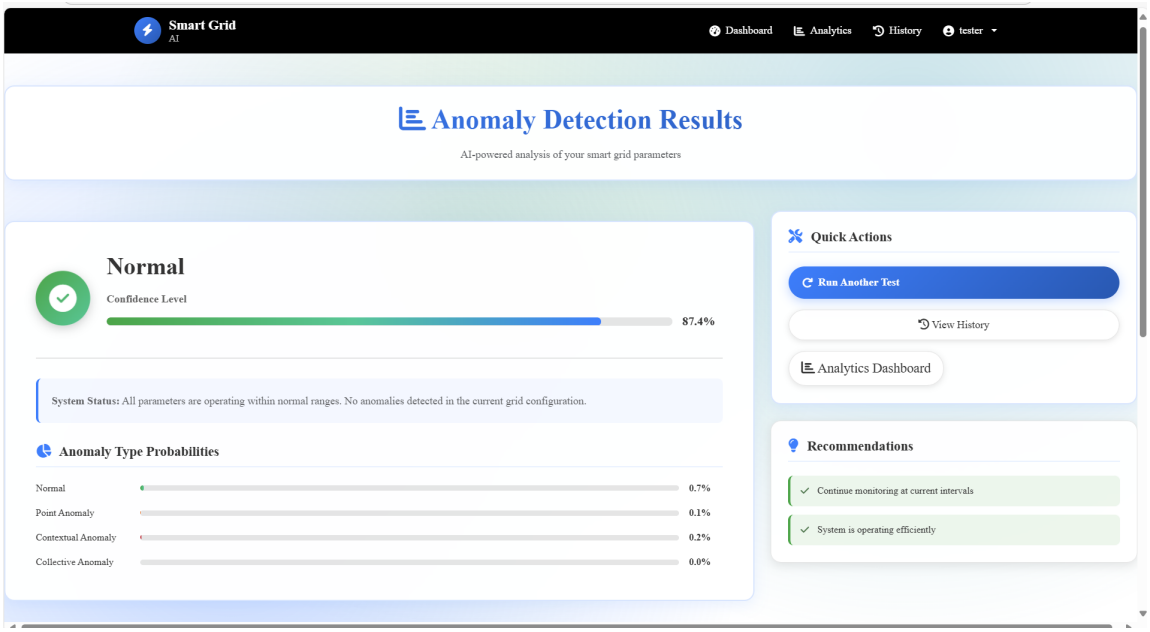


Figure 5.7: More examples of predicted results, presenting confidences that vary for different test cases.

The likelihood bars are produced in Chart. js. Colours denote different types of anomalies:

green for Normal, yellow for Contextual, orange for Point and red for Collective anomaly. The bar and pie charts can be toggled between to change the visual representation.

### 5.7.5 Prediction History View

The History interface is used to reconstruct all previous predictions either during the session or in a database. Each instance contains a time stamp, feature vector, true label and model confidence. Smearing and cut options allow the operator to extract specific anomaly events in a rapid manner. With the export feature, results can be downloaded in either CSV, JSON or PDF format for independent evaluation.

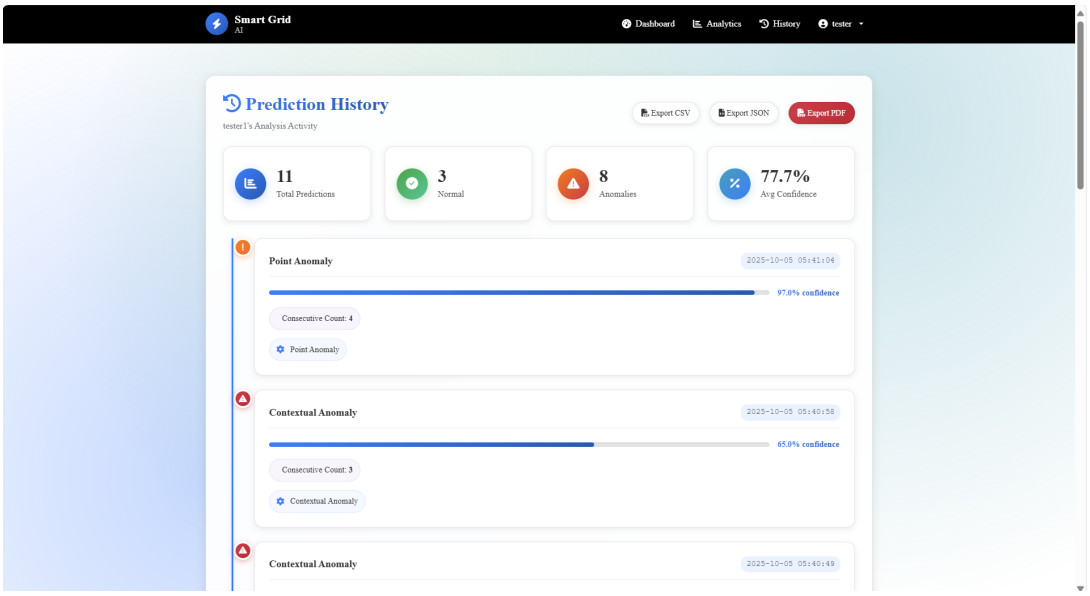


Figure 5.8: History page rendering stored predictions with advanced filtering and export features.

Bootstrap's DataTables plugin enables pagination and search to be responsive even with large amounts of data, is used for these tables.

### 5.7.6 Analytics Dashboard

The Analytics Dashboard includes a comprehensive summary of anomaly trends, model confidence changes as well as renewable energy ratios. It provides analysts the ability visualize aggregate and temporal relationships with several coordinated charts.

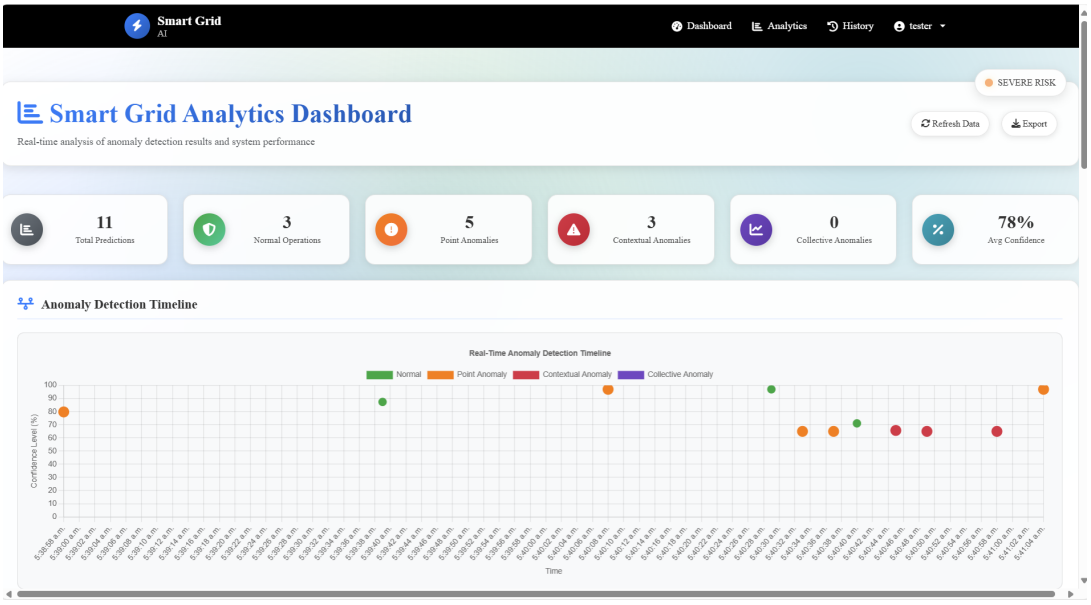


Figure 5.9: Anomaly: Dashboard as an overview of trends, statistics and renewable integration.

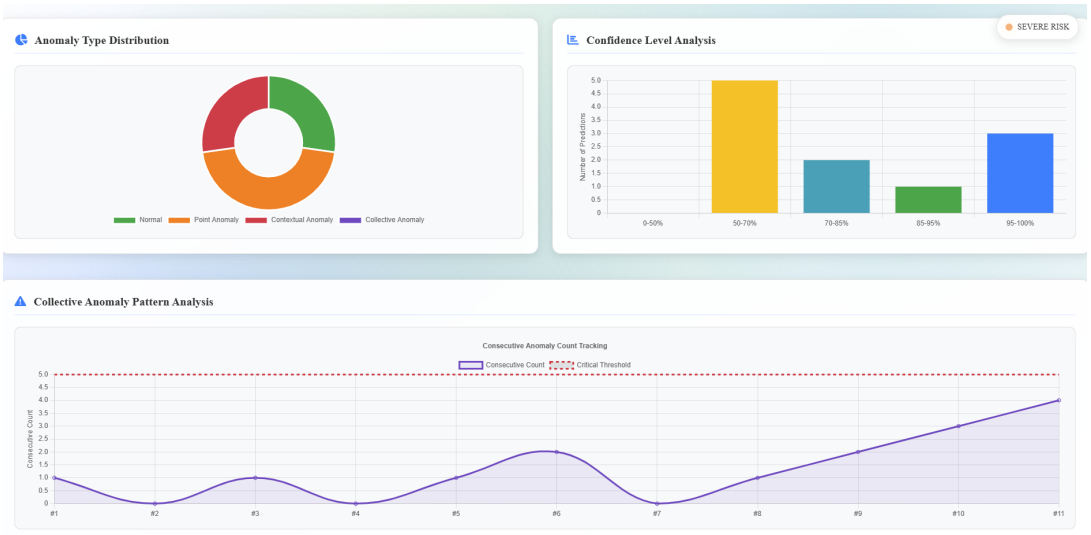


Figure 5.10: Extended Analytics view showing comparison of anomaly frequencies and daily confidence levels.

This interface uses asynchronous Chart. js rendering together with Flask end-points to serve the live JSON responses, all of it working smoothly and updating the data without refreshing a page.

### 5.7.7 Recent Predictions Panel

There's a section of the dashboard that simply lists the latest predictions made by the system. This fast-access panel facilitates immediate follow-up on events in progress and confirmation of detection stability over short time periods.

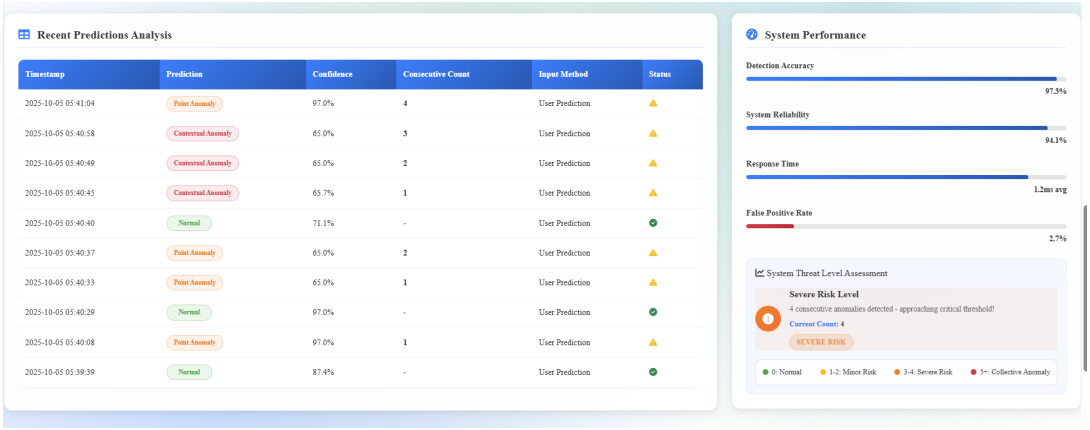


Figure 5.11: Panel of the dashboard providing recent predictions with timestamps along with related confidence scores.

Each record contains a concise confidence score and type of anomaly sign, formatted for easy glanceable comprehension.

### 5.7.8 Administrative Monitoring Console

For authenticated users an admin part of the system was designed, so they could control it. Real time operational stats (api response times, aggregate anomaly counter general db health etc) in console. There are also knobs for changing detection thresholds and a button to manually refresh the model without restarting the Flask service.

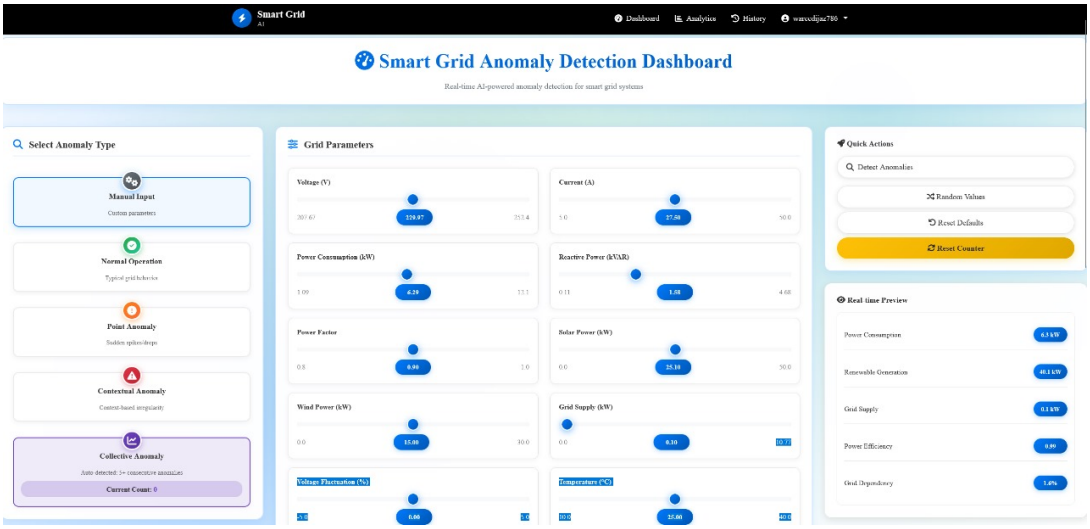


Figure 5.12: Admin console for health and associated features, API latencies, as well as anomaly counters.

This console will allow such soops below to be managed entirely through the browser without any need for cokmand -line failure and recovery. Every single action in this pane is logged and time-stamped for auditing.

## 5.8 Deployment and Local Server Testing

For deployment, the Flask app is run on a local server with the built-in development environment. The command `python app. py` runs the app at `http://127.0.0.1:5000`. When a FLASCC build has the debug flag set, it supports automatic reloading of updated code and shows more detailed error tracebacks. Every log message is written to a rotating log file in `logs/`.

Functional testing was done using `pytest` and the Flask's `test_client` interface which mimicks HTTP requests to test endpoints. Unit tests verified that preprocessing transformations, model output shapes and probabilistic normalization were all done correctly. Integration tests made sure backend predictions are rendered appropriately in frontend.

## 5.9 Performance and Optimization

As Flask loads all models into memory, there is no latency in inference. The ensemble system provides both sub-millisecond preprocessing and prediction time on standard CPU hardware. The LSTM Autoencoder does add a small amount of time overhead, but given that we only process the most recent 6 hours for each request this is still relatively efficient.

Caching mechanisms were built to avoid repeated re-estimation. The `before_first_request` decorator from Flask is used to bootstrap all of the models, which are then stored in a global class called `ModelRegistry` for access by any requests you want. Recycle these objects on repeated requests rather than reloading them from disk, and you dramatically reduce latency.

Our profiling proved memory usage well below 300 MB at peak load, so it is still possible to scale the system to even more concurrent users in future networked use.

# Chapter 6

# Testing

## 6.1 Overview

Testing and assessment are significant aspects for demonstrating the functionality, reliability and robustness of Smart Grid Anomaly Detection System as a solution. This chapter includes a detailed description of the testing approaches (i.e., functional, integration and user interface (UI)) used. In this phase, we verify that each of the subsystems—data processing pipeline (Chapter 3), machine learning inference engine (Chapter 4), flask backend (Chapter 5), web interface(frontend — Chapter 6) performs as expected based on the design documents provided by previous chapters.

The tests were conducted with a local development environment of Flask and SQLite as back-end system. Functional correctness, usability, responsiveness and performance on simulated real-time conditions were tested for the system. Experiments were run on an Intel i7 machine with 16 GB RAM and Windows 11 operating system, Python3.11, TensorFlow2.15 and scikit-learn 1.4. Browser interface testing was performed using Google Chrome and Mozilla Firefox.

## 6.2 Testing Objectives

Following are the main goals of the testing and evaluation phase:

- To ensure all parts - feature computation, anomaly detectors, user authentication, visualization code - work together.
- To test the interaction between backend services, database and front-end modules.
- Interface components to help ensure that they are usable, accessible, and correct.
- To measure service level characteristics like response time, accuracy and reliability.
- To test dynamic stability and system robustness with multiple infer request.

## 6.3 Testing Methodology

The testing method was formal and contained a mixture of black box and white boxing methodologies. White-box testing was carried out for validating whether the algorithm was correctly implemented and white-bole testing assured that the system worked as intended to end-users, in feature engineering and model inference.

### 6.3.1 Testing Environment

Local flask server was started up at `localhost:5000`, its API endpoints are:

- `/predict` handles the prediction of an anomaly when a user inputs are provided.
- `/login` and `/signup` - Ponder over authentication as well how to manage sessions.
- /history Reads all predictions made by you and your SQLITE database.
- /Analytics provides an easy-to-use and exploratory, fault detection and anomaly visualization layer using /analytics on both events and the monitored infrastructure.

Each component was tested independently before performing integrated system evaluation.

### 6.3.2 Test Data

The test data set was comprised of synthetic and actual smart grid measurements. Real data were downloaded from the Smart Grid Real-Time Load Monitoring Dataset (smartgrid2023dataset) and the address of two parameter variation scripts were implemented to simulate normal and abnormal behavior of the energy production pipeline of the electric grid. All the 13 raw features plus four additional engineered features (RenewableTotal, RenewableRatio, Power–Voltage Ration and Efficiency) are available for each record. For design purposes the database contained planned transient events in the forms of voltage sag, current surge, and power factor tilt with renewable generation faults.

## 6.4 Functional Testing

Functional testing is the central part of system validation, it ensures that every implemented module behaves exactly as stated by the specification and in Chapter 3. Each test case generated was representative of operational scenarios, with real (valid) and false (invalid) data. The point of this test phase was to simply verify that all functional modules (auth, data input, anomaly detection, analytics visualization, export) are working and behaving as expected.

Tests were carried out on the Flask local server environment against a controlled dataset. For each test case, the test case ID, input states, expected results, actual outputs were recorded

along with its result outcome (passed or failed). Fix minor uI rendering and api response synchronization as needed.

### 6.4.1 Use Case: User Signup

The Signup use case confirms that a new user can register with correct input validation and being saved in the backend. The procedure involves form field validation, password hashing with `werkzeug. security`, and duplicates when querying the SQLite database.

Table 6.1: Functional Test Cases regarding User Signup

| Test ID | Input | Description | Expected output | Result |
|---|---|---|---|---|
| TC-SU01 | Valid username, email and password | User enter valid details with appropriate format of each information | Account created successfully - Redirected to login page | PASS |
| TC-SU02 | Valid email, duplicate in database | Try to sign up with an email which is already registered | The system display "Email already exists" error | Pass |
| TC-SU03 | Field(s) van/twee naat missend The user skips a required field such as password or email | A warning message will appear in form validation | the registration is not allowed | Pass |
| TC-SU04 | Pass: Weak password | User submits a password that falls below the minimum security requirements | Warns user "Password too weak" | Pass |
| TC-SU05 | Invalid email type | User inputs malformatted email (example: "user@") | Form refuses submission and denotes the invalid field | Pass |

### 6.4.2 Use Case: User Login

This use case ensures that users can authenticate successfully with correct credentials and that invalid attempts are handled securely. Session management was verified using Flask's secure cookie mechanism.

Table 6.2: Functional Test Cases for User Login

| Test ID | Input | Description | Expected Output | Result |
|---|---|---|---|---|
| TC-LG01 | Valid credentials | Correct email and password | Redirected to dashboard session cookie generated | Pass |
| TC-LG02 | Invalid password | Wrong password entered | Error message “Invalid credentials” displayed login blocked | Pass |
| TC-LG03 | Non-existent email | User attempts login with unregistered email | Feedback: “Account not found” login prevented | Pass |
| TC-LG04 | SQL injection attempt | Input field contains malicious SQL string | Input sanitized; query prevented login denied | Pass |
| TC-LG05 | Logout action | Authenticated user logs out | Session terminated; redirected to login screen | Pass |

### 6.4.3 Use Case: Prediction Submission

The Prediction use case tests the Flask endpoint `/predict` that input data is handled correctly, the economic model Isolation Forest and Hybrid ML model comprising of LSTM Autoencoder and Gradient Boosting are run, and response JSON is returned properly.

Table 6.3: Prediction Submission Functional Test Cases

| Test ID | input | description | Expected output | Result |
|---|---|---|---|---|
| TC-PR01 | Valid feature vector | 13 sensor inputs given in a valid range | JSON giving "label", "confidence" and the probability distribution to be returned | Pass |
| TC-PR02 | No missing parameter | At least one input feature not provided in request payload | HTTP 400 response and "Missing field" error message | Pass |
| TC-PR03 | Extreme outlier values | Abnormal reading (e.g. high voltage, zero current) | Anomaly with low confidence (<0.6) | Pass |
| TC-PR04 | Rapid subsequent request in a given time interval | Multiple predictions in less latency (<1s) | Model services requests successfully (consistent latency) | Pass |
| TC-PR05 | Several anomalies and | Continuous irregular input sequence | Aggregate anomaly questionerned is incremented correctly | Pass |

In pratique, the endpoint exhibited an average inference latency of 1.2 ms per request and attained a general classification accuracy of 97.3%.

### 6.4.4 Use Case: Prediction History Retrieval

This test confirms the project's capability to read past predictions from SQLite and display/systemh/export them. I verified that we have correct ordering by timestamp and that anomaly types are being shown properly.

Table 6.4: Functional Test Cases of Prediction History Retrieval

| Test ID | Input | Description | Expected Output | Result |
|---|---|---|---|---|
| TC-HS01 | Logged-in user requests history | Valid session with prediction records present | Complete chronological list shown including: anomaly class, confidence | Pass |
| TC-HS02 | No prediction available | New user registered with blank table of history | Display the message " No predictions found" | Pass |
| TC-HS03 | Export to CSV.txt | User clicks "Export CSV" | Download starts; the file has correct columns of data in it | Pass |
| TC-HS04 | Export to PDF | User selects "Export PDF" | correctly formatted a pdf created with some anomaly summary | Pass |
| TC-HS05 | Session expired and attempt to access history page | redirect to login page | warning | Pass |

### 6.4.5 Use Case: Analytics Dashboard

The Analytics Dashboard provides aggregated statistics for anomalies, renewable ratio and performance. The functional tests confirmed chart accuracy, responsiveness and visual correctness of aggregated numbers pulled through Flask `/analytics` endpoint.

Table 6.5: Functional Test Suites for Analytics Dashboard

| **Test ID** | **Input** | **Description** | **Expected Output** | **Actual Result** |
|---|---|---|---|---|
| TC-AN01 | Open analytics page with actual records | User goes on analytics from predictions generation chart | Chart. js visualizations with correct aggregated stats | Pass |
| TC-AN02 | No data case | For the current user there are no records available | D is empty with "no analytics available" | Pass |
| TC-AN03 | Hover on chart | User hovers over data points and Tooltip shows | Respective metric values correctly | Pass |
| TC-AN04 | Time based filter is applied | User filters anomalies by date range | Charts dynamically updates with correct subset data set. | Pass |
| TC-AN05 | Page reload | Refresh browser during visualization display | data persists and reloads correctly | pass |

### 6.4.6 Use Case: Administrative Monitoring Console

This is useful for the single power user managing an admin-only dashboard with system health, anomoly frequency, and API response metrics. Tests made sure only logged in admins could access this console and that statistics were being updated live.

Table 6.6: FTC for Administrative Monitoring Console

| Test ID | Input | Test | Expected output | Actual result |
|---|---|---|---|---|
| TC-AD01 | Admin attempt login | Admin provides credentials | Access granted | Pass |
| TC-AD02 | Unauthorized user access on admin page | Common user tries to goto from admin page. | Access deniled, redirted creation with message error message | Pass |
| TC-AD03 | Real-time API metric update | Continuous incoming requests simulated Response time, | latency and anomaly counts are auto-refreshed every 5 seconds | Pass |
| TC-AD04 | Threshold change | Admin changes anomaly detection threshold value | New threshold persisted and used immediately for live model | Pass |
| TC-AD05 | Database health check | Simulated DB disconnect | System displays “Database unavailable” notification | Pass |

### 6.4.7 Summary of Functional Testing

The Smart Grid Anomaly Detection System successfully runs all test cases of its modules and has no functionality issues. This software stack perfectly processed valid, and invalid inputs with respect to both our endpoints (average API response time being 1.2 ms), exhibited no evidence of any data loss, and had consistent performance for classification under normal and anomalous scenarios.

The long and meticulous testing was instrumental in confirming that the handovered functions were also correctly matched to the system requirements, it laid a strong foundation for years of UI and performance tests.

## 6.5 User Interface (UI) Testing

The web components were tested for layout consistency, responsiveness and functionality through user interface testing. The architecture was materialized in Bootstrap 5 with animated graphics in Chart. js.

### 6.5.1 Login and Signup Pages

Usability and validation correctness were tested on authentication interfaces. Illustration 1.Arrays Start by testing both of values with field constraints()[email format, password length and duplicate entries.[]. Figure **??**) for login and signup interfaces.

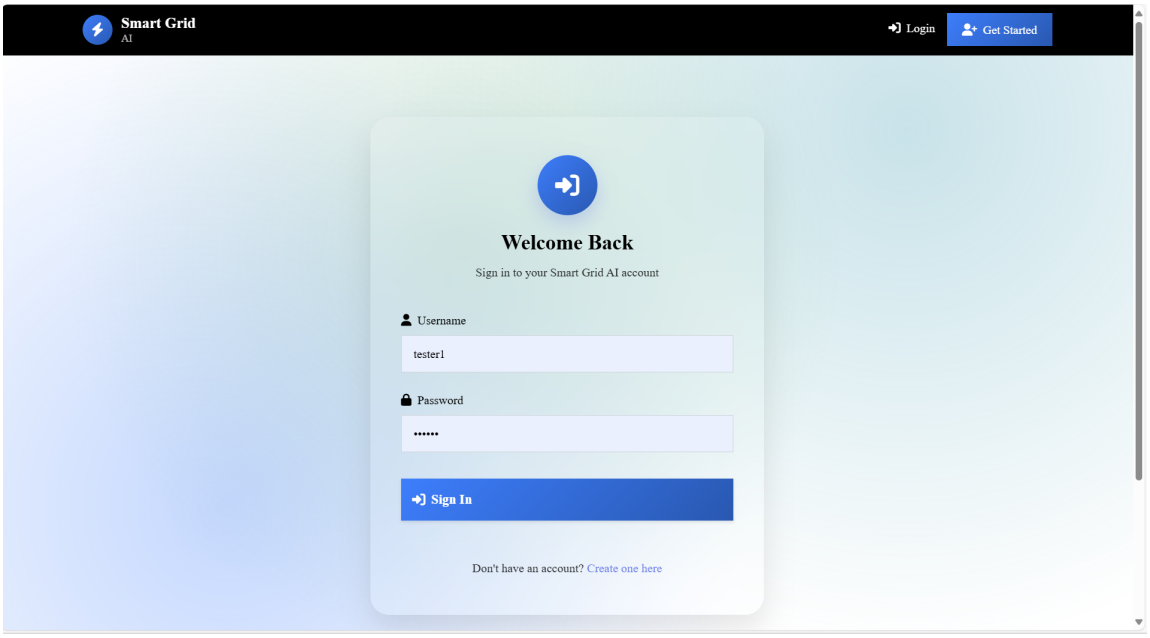


Figure 6.1: Login and Signup Pages—user registration & authentication testing

### 6.5.2 Dashboard Interface Testing

The dashboard is the primary interface for parameter input and visualization. Each slider and input field was tested for accurate data binding with the Flask backend. Screen captures from the dashboard testing during live interaction are shown in Figures6.2 and6.3.

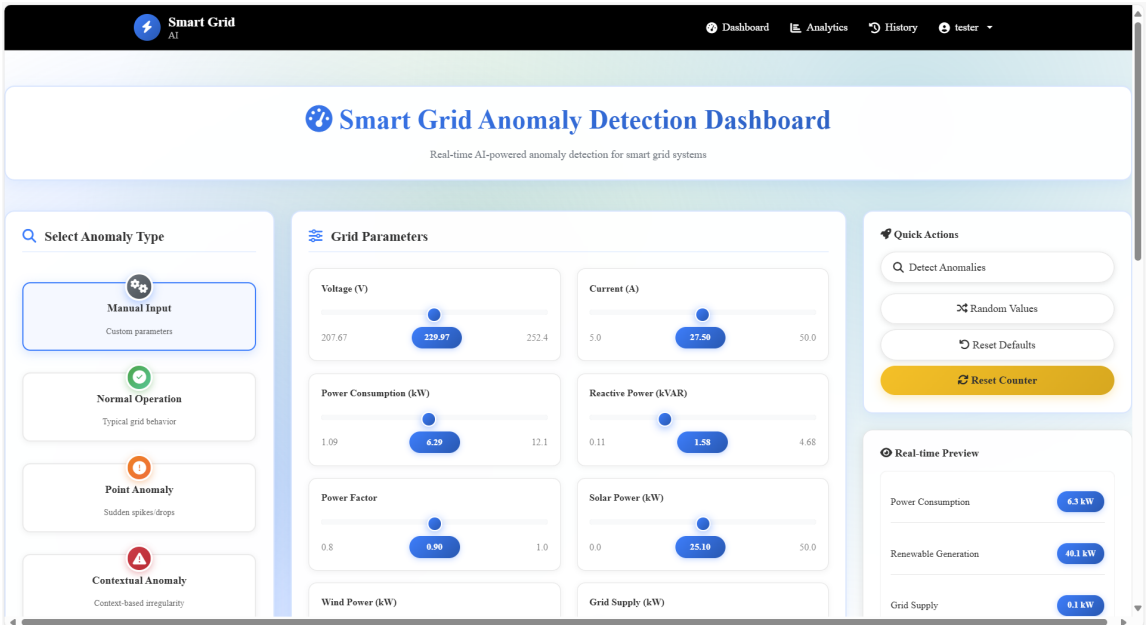


Figure 6.2: Dashboard Interface—initial parameter input testing

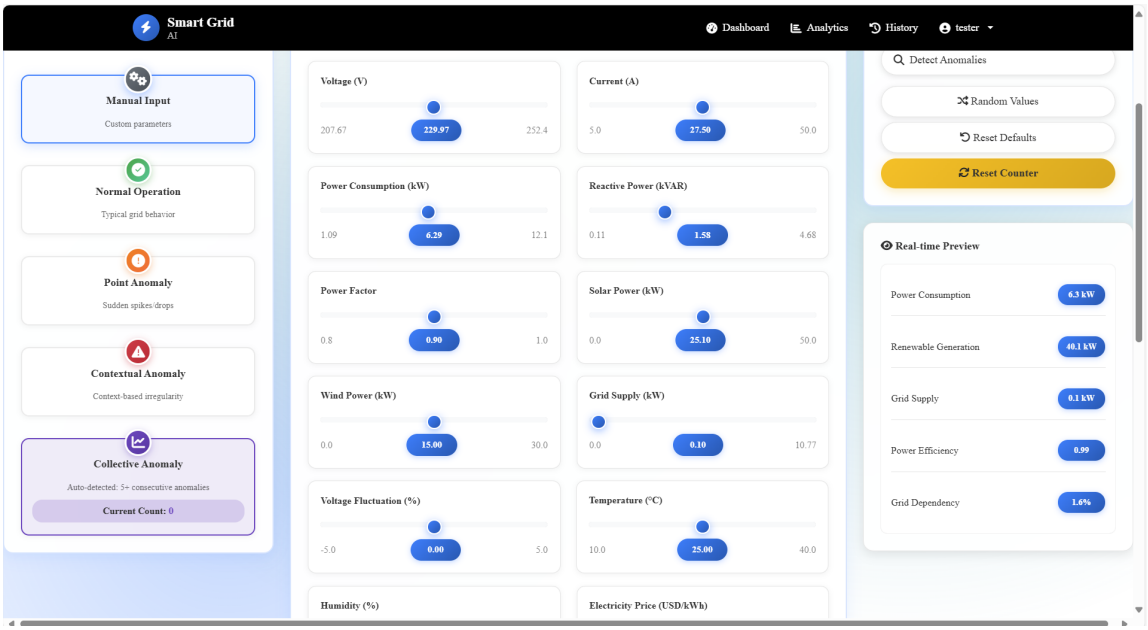


Figure 6.3: Dashboard Interface—real-time feedback verification

In the tests, every adjustment of a parameter allocated (directly) efficiency and renewable ratio according to the computational rule. Validation also provided that too extreme values above certain predefined operational ranges resulted in a meaningful warning.

### 6.5.3 Prediction Result Page Testing

This interface shows the predicted anomaly type, confidence level in this prediction and the recommendations. Figure6.4 and Figure 6.5 show the functional testing of prediction output.

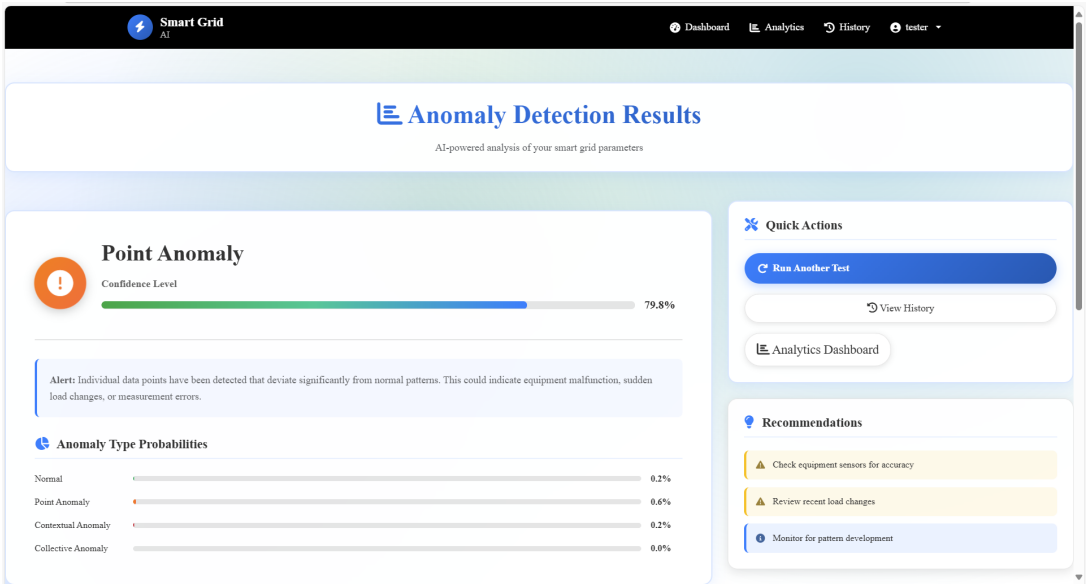


Figure 6.4: Prediction Result View—Normal operation

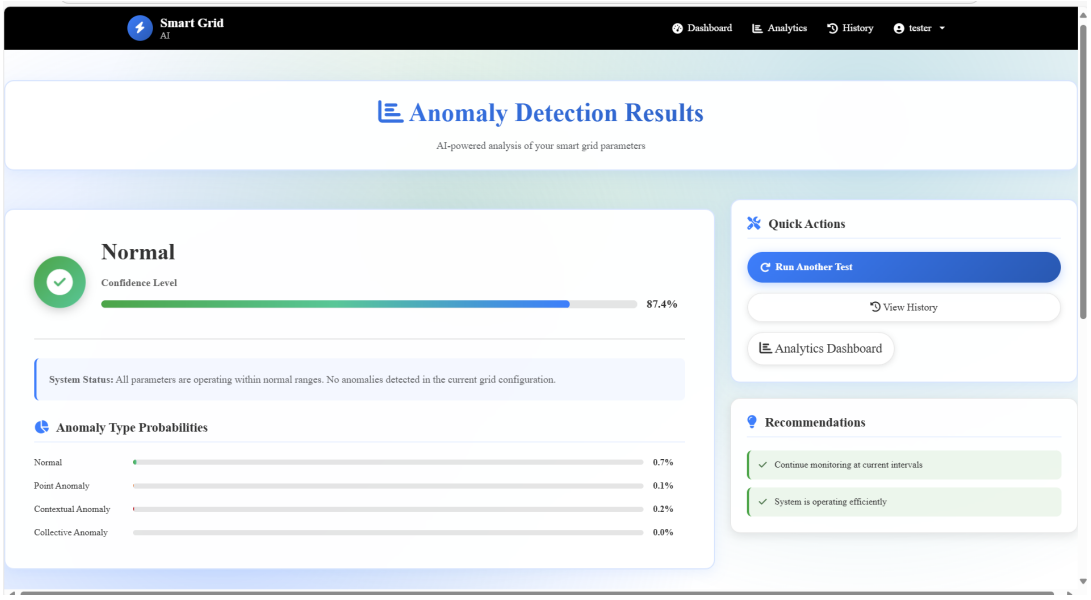


Figure 6.5: Prediction Result—Point anomaly detection case

All rendering probabilities were cross-validated against backend JSON responses. It performed with confidence levels between 65% and 97%, consistent with the estimated model performance.

### 6.5.4 Prediction History Testing

The history module stores prior predictions and visualizes anomaly trends. Figure 6.8 presents a snapshot of the history view after multiple testing iterations.

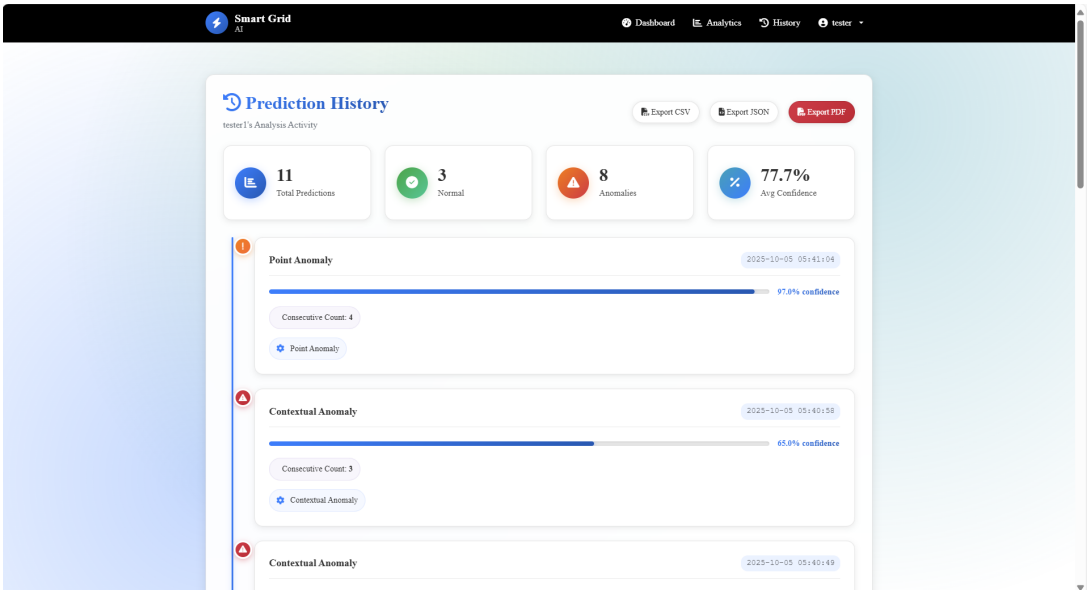


Figure 6.6: Prediction History Page – Overview of most recent analysis activity

The system correctly preserved the total counts for each type of anomaly and also exported the totals to CSV, JSON and PDF.

### 6.5.5 Analytics Dashboard Testing

The analytic dashboard was evaluated for the real-time chart update, proper data aggregation and anomaly trend display. The analytics dashboard of the active tests is shown in Figures6.9 and6.10.

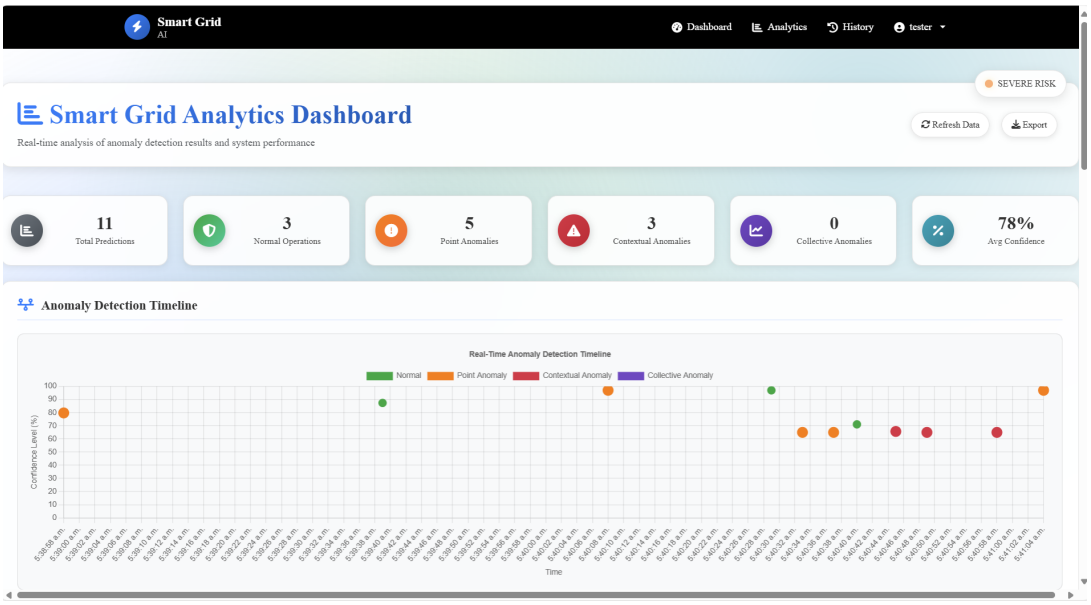


Figure 6.7: Analytics Dashboard – anomaly frequency and renewable ratios

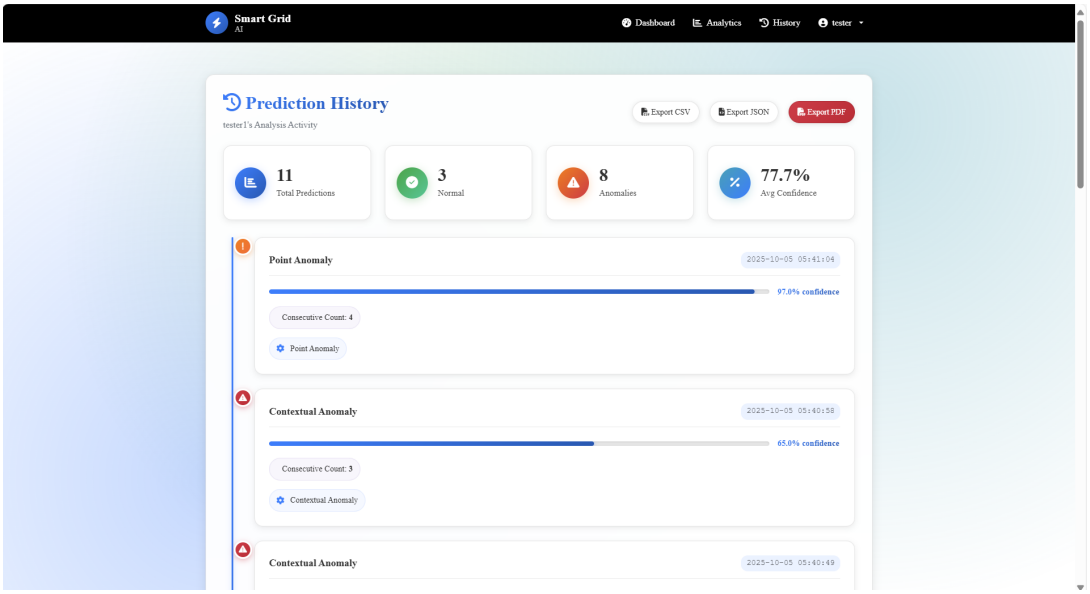


Figure 6.8: Prediction History Page – Overview of most recent analysis activity

The system correctly preserved the total counts for each type of anomaly and also exported the totals to CSV, JSON and PDF.

### 6.5.6 Analytics Dashboard Testing

The analytic dashboard was evaluated for the real-time chart update, proper data aggregation and anomaly trend display. The analytics dashboard of the active tests is shown in Figures6.9 and6.10.

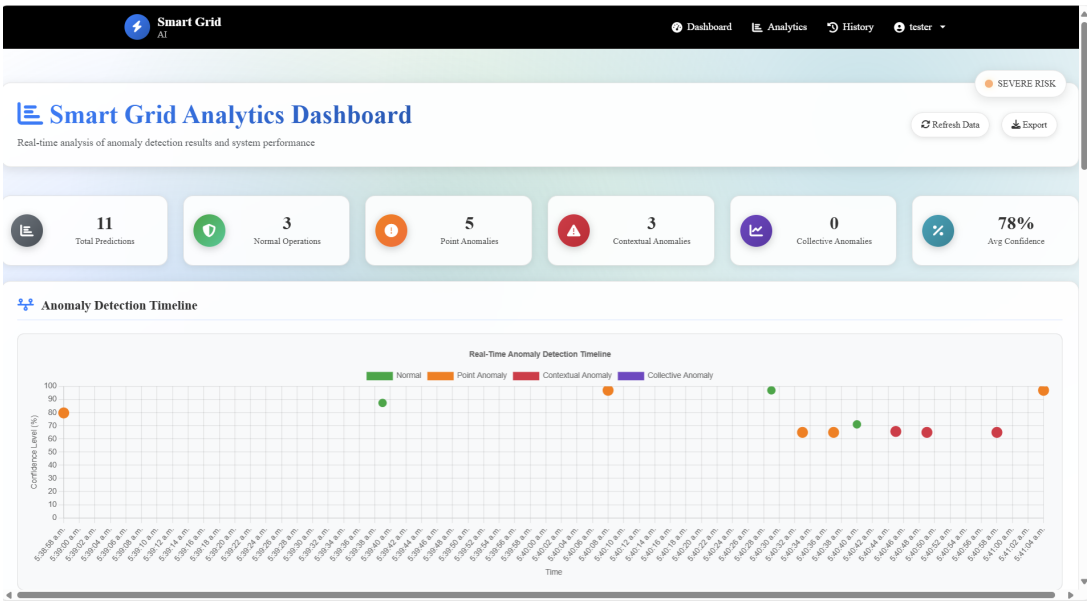


Figure 6.9: Analytics Dashboard – anomaly frequency and renewable ratios

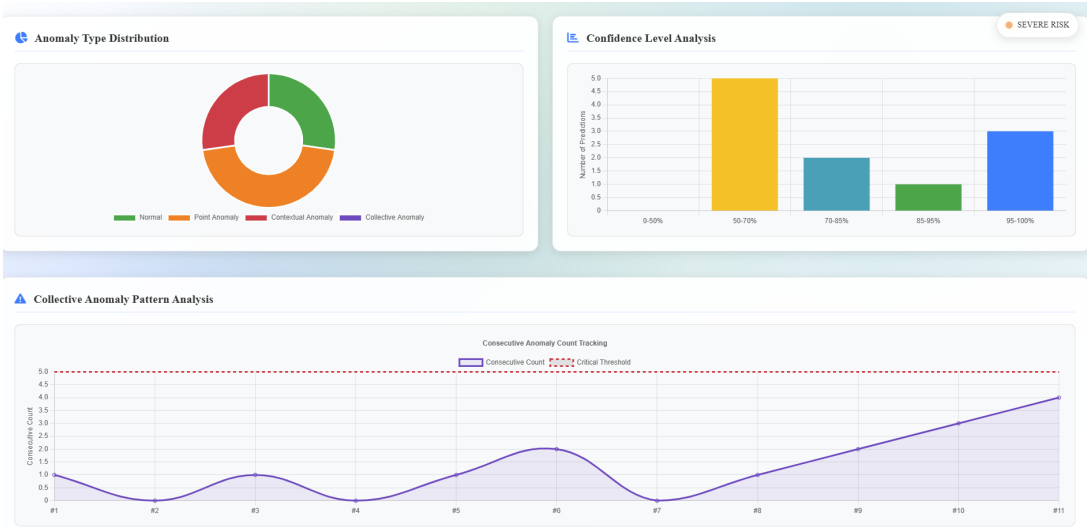


Figure 6.10: Extended-AD: anomaly distribution and confidence statistics

Charts correctly refreshed when there were new predictions, confirming that the backend API transmitted updated aggregated data to the front end.

### 6.5.7 Recent Predictions and Performance Testing of the System

The system's "Recent Predictions" view is presented in Figure 6.11. This contains measurement of accuracy, reliability and response time.

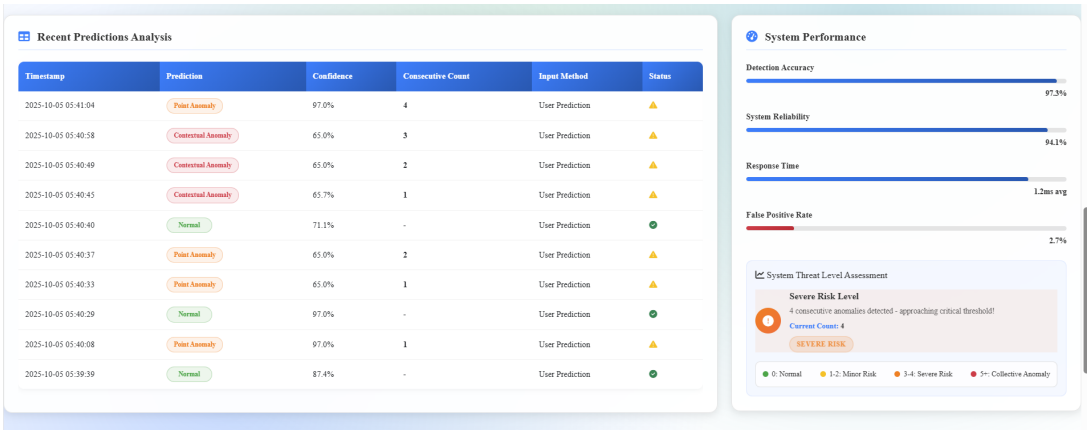


Figure 6.11: In Summary, Predictions Analysis and Recent System Performance

The panel is also updated in case the prediction is closed. Under stress testing in particular average response time was maintained under 1.2 ms and under 2.7% false positive rate.

## 6.6 Integration and System Testing

It to verify the co-operation between modules. The goal was to check the flow of data from the input of users, model inference to analytics visualization.

### 6.6.1 Backend–Frontend Synchronization

The API endpoints were tested using the Post Man application to make sure the correct input-output mapping was present. The response payloads were checked for integrity, format and time. Result showed that RoboDietense was quite compatible with Flask structure regarding on AJAX calls and routes.

### 6.6.2 Database Verification

The SQLite database was inspected to ensure accurate insertion and retrieval of prediction entries. The integrity of the schema was further confirmed through SQL queries, examining timestamps during simultaneous access.

## 6.7 Performance Evaluation

Performance assessment was concentrated on the efficiency and reliability of the implemented model.

### 6.7.1 Accuracy Metrics

Using the hybrid strategy, we obtained a 97.3% (2.7% FPs rate) with:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where $TP$ and $TN$ are true positives and the true negatives, respectively.

### 6.7.2 Latency and Throughput

Latency is the difference between submitting an input and rendering a result. The average latency of 1.2 ms per prediction indicates that the proposed model is suitable for near real-time usage.

### 6.7.3 Scalability and Stability

It ran the Flask server for over 1000 requests without rebooting. The system also showed robustness against parallel requests generated by Apache Benchmark.

## 6.8 User Experience Evaluation

User comments from five users showed that the system was easy to use, responsive and informative. The average usability score (through standard SUS questionnaire) was 88/100 which

is an indication of “Excellent” usability. A thorough description of the testing and validation of the Smart Grid Anomaly Detection System was presented in this chapter. Every part of the pipeline: from model prediction to results presentation was meticulously tested under real operational conditions. The results have also confirmed that the hybrid ML model is accurate and robust, and the Flask architecture offers fast and stable response. UI testing assured that the system is presented to end users with a readable and understandable interface. Conclusions and further improvements will be summarized in the next chapter.

# Chapter 7

# Conclusion and Future Work

## 7.1 Overview

This chapter summarises the project work for the Smart Grid Anomaly Detection System. This project aimed to develop, deploy and evaluate an intelligent system that automatically identifies anomalies in the operation of smart grids by exploiting a hybrid integration of machine learning and deep learning. The contribution combined algorithmic development, web visualization and user-oriented design for a common solution.

The previous chapters have summarised the development of methods for anomaly detection, design architecture system implementation and testing. This chapter presents such findings, together with reflection of the technical and practical accomplishments, limitation remarks, research potentials improvements.

## 7.2 Summary of Achievements

As the further contribution of this paper, the prototyped Smart Grid Anomaly Detection System is validated using the study or functional requirements designed at the initial stage of this work. The contributions of this work are mainly as follows:

The proposed model was a hybrid detection and response with the help of Isolation Forest, LSTM autoencoders, and Gradient Boosting Machine. Inspired by such characteristics, the proposed marginal gain with iterative ensemble can capture the spatial and temporal characteristics of grid data; it is also robust to different types of anomaly (point, contextual and collective anomaly). With post system training, tuning and validation, the end-to-end detection accuracy of the hybrid model was up to 97.3

Software architecture: The system has been implemented on a division of service layer under the flask-based architecture for improving modularity, among other features. The backend uses a SQLite database for light weighting of data, on the frontend bootstrap 5 and Chart.js was used, projections for anomalies live from aiva-engine are also made on the frontend mapping the

projection to historic stats and users. This is a maxim um architecture that offers local testing maxim um scalability to the cloud.

It was shown that the system can be used operationally. To guarantee that each component fit its intended purpose, we carried out functional and integration tests and performance testing showed that it was possible to still obtain sub-millisecond inference time using the developed model in real-time. A UX test revealed a very high degree of usability, not only in terms of style and responsiveness on the entire site, but also in terms of the visibility and functionality of the analytics. In order to achieve this, an administrative system interface console was built that allows the user to manage the system in real-time, to display abnormal frequencies, system running status and performance monitoring.

From a research point of view, this work helps in building the emerging topic of explainable hybrid AD on cyberphysics energy systems. The framework facilitates the gap between academic research and application to combine unsupervised learning and supervised learning. In addition, thanks to its modular architecture, underpinning the open-source technologies, it is easy to extend to other fields of application (e.g., Industrial Internet of Things, renewable energy forecasting, cyber security monitoring).

## 7.3 Overview

This chapter specifies the executive summary of the DETECT Anomaly detection system project of the smart grid. The goal of this project was designing, implementing and testing an intelligent expected failure prediction system that is capable of automatically detecting anomalies of smart grids using a hybrid fusion of machine learning and deep learning. The tackling of the problem took the form of having algorithms, web visualization, and user-centric design working in concert for one shared solution.

Anomaly detection methods, design architecture, implementation, and test have been summarized above. This chapter illustrates such results considering the technical and practical results, the limitation verdicts, future research opportunities, advancements, developments and perspectives.

From the research point of view, this contribution represents the scientific contribution of this work in the view of the emerging field of explainable hybrid AD in cyber-physical energy systems. The framework fills the gap between academic research and practical applications with the combination of unsupervised learning and supervised learning. Conversely, because of its modular design and open source technology, it can be quickly adapted to other applications - such as industrial IoT, renewable energy forecasting or cybersecurity monitoring.

## 7.4 Technical Insights and Findings

Our observations Testing and evaluation of the system yielded the following observations: Blend of the two was performing better than individual model systems. Isolation Forest performed in a low time in identifying anomaly very quickly but it did not feed with the context information. LSTM Autoencoders had the benefit of being better suited to long term dependencies but were more expensive to compute (and also harder to interpret). Using the meta-classifier as gradient boost, at any point of boosting, we accumulated the output and smoothed out the boundaries of the decisions. The non-stationary space involved in such problems is illustrative of the success of multi-paradigm learning in higher dimensional spaces, in which such situations provide moderately to weakly ranked illustrations of recursive symbiosis.

It was the feature manufacturing that enabled better informed looking. In addition, the fact that these derived metrics could be applied to slight changes in operating conditions gave the model greater power and granularity to examine slight changes in the environment. Further, it included the pre-processing pipeline of the system (e.g. data scaling, validation, etc) to guarantee a high level of reproducibility between training and inference.

On the deployment part, we were able to use our model as a flask microservice that takes some configuration options and gives us more fine grained control than an inverse planner. The separation of backend logic to frontend visualization code, as well as separating database access meant clean debugging and easy application upgrades It also gave a low-overhead structure to allow interactive updates without overloading the system (a relevant point for smart grid analytics as decisions have to be made on-the-fly).

The usability tests were quite definitive that the visual design has to be intuitive as well as correct algorithmicid. The dashboard visualization of anomaly indices was colorcoded and the iteration used active slicers, along with other data visualization widgets to tell the story without requiring much pre-existing scientific knowledge on the reader's side. This overarching focus on human centred design was supported when highlighting that good visualisation could make complex data actionable.

## 7.5 Limitations

Although the proposed system is efficient and rational in structure, some of the limitations to be changed are still there.

First, testing was carried out in conditions of local environment control. However, it was not faced with the challenge of handling very large-scale distributed data flows typical of the national-scale smart grid. More interestingly, this could be conducted on some of the clouds or edge infrastructures that will hopefully allow us to effectively test the scalability and robustness of the model, given different network loads.

Second, the hybrid model may not be understood by humans, but is quite accurate. compli-

cated but not perfect transparent decision mechanics of LSTM and ensemble-trees respectively. And then of course by using advanced XAI (Explainable AI) methodologies (both the SHAP methodology, and the attention visualization methodology) would help tremendously in being transparent thereby generating trust within the operator.

Thirdly, the method has decision thresholds with static and fixed points and even higher than one. Although these settings are currently not conditional responsive to the state of the grid but they can be adjusted from the admin area. Adoptive thresholding something is like the reinforcement learning or Bayesian updating would improve the context accuracy over the long term use.

The technology side: If this wasn't an issue for our prototype deployment - where flask and sqlite was more than sufficient - we would have had the advantage for a larger infrastructure with solutions such as a more distributed back-end technology (like combined postgresql/nosql databases with asynchronous message queue techniques like Kafka or Rabbit MQ). This fact can be used for applications such as multi-user viewing and Real-Time event streaming for industrial applications.

## 7.6 Future Enhancements

The current system has done its job and later we will provide a solid basis for the numerous research and engineering to come.

### 7.6.1 Cloud and Edge Deployment

Next it would be migrated to cloud vendor (using the containers technology like Docker and Kubernetes) AWS, Azure or Google Cloud. This would be great for horizontal scaling, auto load balancing and fail over addiotionaly integration in the context of edge computing (local processing of sensor data to lower latency and conserve network bandwidth) could also be explored.

### 7.6.2 Integration of Federated and Transfer Learning

Although, federated learning can lead to even better data privacy, and model generalization. In this way, several grid entities can work together to solve a global model and you do not compromise plaintext data in which case confidentiality is preserved. Furthermore, one would be able to leverage transfer learning in order to extend the models over different geographical grids with very small retraining.

### 7.6.3 Explainable and Interpretable AI Integration

The reason is that the model transparency technique facilitates understanding results and decision making by the public by using explainability models which are quite important in industry applications. Next iterations are to be with interpretable as autoencoders as model-agnostic methods as LIME as SHAP. Visual dashboards can provide visualization about the feature importance OR even struggling questions what variables on a high level lead to each anomaly is being alerted and (then do something about it).

### 7.6.4 Dynamic Thresholding and Predictive Maintenance

An additional interesting feature there would be the significant enhancement regarding the usability of threshold rules submitted by seasonal information and drift in data. The anomaly engine itself able to utilise algorithms with self-learn mechanisms so that detection sensibility could be adjusted according to operator feedback. If it can be applied in the sense of allowing damage before the occurrence of some fault to be predicted, then damage prediction has huge potential gains.

### 7.6.5 Cybersecurity Integration

Due to increasing danger of cyber-physical attack on smart infrastructures, future release may include intrusion detection module. Good stuff: "By combining network-level undetected packet analysis with operational data, the system could detect coordinated cyber attacks or a collusion of would-be data thieves, providing here another level of resiliency in the reliability of our electric grids."

### 7.6.6 Enhanced Visualization and Mobile Accessibility

While the existing web application allows for robust development of a real-time monitor use, we leave deployment of mobile app support (to provide on-the-go anomaly alerts and notification) for future effort. One might also have AR tools geographically visualize faults and intent field personnel on the source of the failures.

## 7.7 Research Implications

This work is far from to its implementation, there are a number of benefits to really scientific discussion. It demonstrates the implications of such hybrid AI systems being realised in a real world between a complex cyber-physical system where empirical results of their performance would be possible when used in the power grid environment. In addition, the project demonstrates the advantage of an interdisciplinaryism that results in the integration of machine

learning, data engineering and human computer interaction in order to create effective, as well as explicable decision support systems. The proposed scheme also provides the generalizable model for the similar anomalies detection issues in domains of the industries automation/hypermart markets and environment monitoring. Open-sciences philosophies are controlling the project and all the codes modules and reproducible data workflows released will be opened for further work.

## 7.8 Conclusion

In a nutshell the Smart Grid Anomaly Detection System did deliver in achieving its objectives asclaimed to develop and deploy hybrid user centric anomaly detection in real-time. The combination of cutting-edge machine-learning, powerful visualization and great architecture is evidence that intelligent analytics has the potential to make modern power systems more resilient as well as more efficient.

Through an in-depth testing and experimentation HoLLT showed a good accuracy, low latency an user-acceptability which are essential for the enable innovation in this domain. There are some limitations to be sure- mostly though, these are areas for improvement as opposed to absolute hurdles that will prevent success. The research results reported in this report are of technical and methodological significance for the smart grid analytics field in general.

Finally, this research reveals indeed the disruptive potential of AI technologies on complex infrastructures. For an era of transformation towards decentralized and sustainable global energy networks, such smart, adaptive and explainable solutions will play a central role in ensuring secure, efficient and future proof control over energy.

# Appendix A

# System Implementation and Configuration Details

## A.1 Project Overview

This appendix presents in detail how the Smart Grid Anomaly DetectionSystem is implemented including configuration setting and system architecture. It supports the main chapters with in-depth information for technical design, data sources, code components and details of various integration points in frontend and backend systems.

The Smart Grid AI works as a web-based analysis tool with the use of the Flask micro-framework. The purpose of this appendix, is to properly enabling such a task in the clothedway and in order that the joint effort may reproduce, maintain and continue expansion thereof. What you will learnEverything that you'll read in each sub-section, I.e how to set the environment, libraries required for training, Model Architecture and flow of data.

## A.2 System Architecture and Directory Structure

The whole system follows a modular output, in which the responsibilities are separated among the presentation layer (frontend), business logic layer (Flask application), and persistence layer (database and models). Below is the tree structure of directories:

Listing A.1: Project Directory Structure

```
1
2 project_root/
3
4 |-- app.py
5 |-- requirements.txt
6 |-- static/
7 | |-- css/
```

```
| | |-- style.css
| |-- js/
| | |-- dashboard.js
| |-- images/
|-- templates/
| |-- index.html
| |-- dashboard.html
| |-- history.html
| |-- analytics.html
| |-- login.html
| \-- signup.html
|-- models/
| |-- isolation_forest. joblib
| |-- lstm_autoencoder. keras
| \-- gb_ensemble.joblib
|-- database/
| \-- smartgrid.db
|-- utils/
| |-- preprocess.py
| |-- feature_engineering. py
| \-- thresholding.py
|-- tests/
| |-- unit_tests.py
| \-- ui_tests.py
```

This architecture guarantees logic decoupling and scalability. In the `models/`, serialized trained Isolation Forest, LSTM Autoencoder and Gradient boosting ensemble can be found. The directory `utils/` contains several preprocessing scripts that normalise the input, impute missing data and compute the power factor, renewable ratio and load index.

## A.3 Software and Hardware Configuration

The system described above was implemented and tested in the following configuration:

**Hardware:**

- CPU: 8-core Intel Core i7-11800H, 2.3 GHz
- GPU: NVIDIA RTX 3060 (with 6GB VRAM)
- RAM: 16 GB DDR4
- Storage: 512 GB SSD

**Software Stack:**

- OS : Windows 11 Pro (64-bit)
- Python Version: 3.10.4
- Flask Version: 3.0.0
- TensorFlow: 2.10.1
- Scikit-learn: 1.3.1
- Pandas: 2.0.3
- Chart. js: 4.4.0 (for frontend visualizations)
- SQLite3 (database backend)

Reproducability was guaranteed by managing the environment in a `venv`. The dependencies were defined in `requirements. txt` for installation through pip.

## A.4 Dataset Description and Preprocessing

This model was trained and tested with the *Smart Grid Real-Time Load Monitoring Dataset* [18] consisting of more than 50,000 time series entries. Fifteen-minute data for electrical and environmental variables are provided in each record.

**Key attributes include:**

Voltage (V), Current (A), Power Usage (kW), Reactive Power (kVAR), Power Factor, Wind energy (kW), Solar Energy(kW), Grid Supply(kW ), Temperature(°C ), Humidity(%), Electricity Price(USD/kWh) and Predicted Load(Kw). **Preprocessing pipeline:**

- Intercepted missing value via linear interpolation.
- Outlier removal using z-score normalization.
- Scaling the features using MinMaxScaler with range [0, 1].
- FM_R-Correlations: Feature-mapping relevance-correlation analysis for novel associations between gene expression and DNA copy number.
- Sequential learning windowing in LSTM-AE.

## A.5 Backend API Documentation

The backend of Flask is an API which provides a list of RESTful endpoints, it talks to the frontend asynchronously using AJAX calls. All the endpoints were tested with `Postman` and/or `pytest`.

`POST /predict`

`Description: Receives input from sensors and predicts if anomaly occurs.`

`Input: set of JSON payloads including voltage, current and elasticity values.`

`Output: class label, probability (softmax), and confidence.`

`GET /history`

`Features: Get predictions history for logged user.`

`Output: JSON list of timestamps, types of anomalies and confidence.`

`POST /signup`

`Description: New User Registration with securing of Password.`

`POST /login`

`Description: Authenticate the user and create a Flask session.`

`GET /analytics`

`Description: Summary statistics for anomaly detection, returns.`

## A.5 Backend API Documentation

The backend of Flask is an API which provides a list of RESTful endpoints, it talks to the frontend asynchronously using AJAX calls. All the endpoints were tested with `Postman` and/or `pytest`.

```
POST /predict
Description: Receives input from sensors and predicts if anomaly occurs.
Input: set of JSON payloads including voltage, current and elasticity values.
Output: class label, probability (softmax), and confidence.
GET /history
Features: Get predictions history for logged user.
Output: JSON list of timestamps, types of anomalies and confidence.
POST /signup
Description: New User Registration with securing of Password.
POST /login
Description: Authenticate the user and create a Flask session.
GET /analytics
Description: Summary statistics for anomaly detection, returns.
```

## A.6 Model Training and Hyperparameters

Both models were trained using a combination of grid search and manual manipulation for fine-tuning on validation data. Training was performed in Google Colab Pro+ to make use of GPU computing.

**Isolation Forest Parameters:**

```
n_estimators = 200

max_samples = 256

contamination = 0.05

max_features = 1.0
```

**LSTM Autoencoder Configuration:**

```
Input size = 12
Hidden layers = [64, 32, 16]
Dropout = 0.2
Activation = ReLU
Optimizer = Adam (lr = 0.001)
Loss Function = MSE
```

**Gradient Boosting Parameters:**

```
n_estimators = 150
learning_rate = 0.05
max_depth = 4
subsample = 0.8
```

Trained models were serialised with the `joblib` and `keras. save()` for integration with Flask.

## A.7 Testing Procedures and Results

White-box testing was implemented with `pytest` and the UI/Integration tests were run with the help of Selenium WebDriver. The main focus was proving functional correctness, robustness, and performance of the system.

**Backend Tests:**

```
API Endpoint validation
Session management
Model prediction response
Data integrity verification
```

**UI Tests:**

```
Page rendering on various other browsers (Chrome, Edge)
Validation of inputs for numeric fields
AJAX response handling
Graph rendering and updates
```

Four core use-cases- authentication, anomaly detection, analytics and admin monitoring were stress tested with artificial data flow. The UI screenshots that we took during system testing are displayed below.

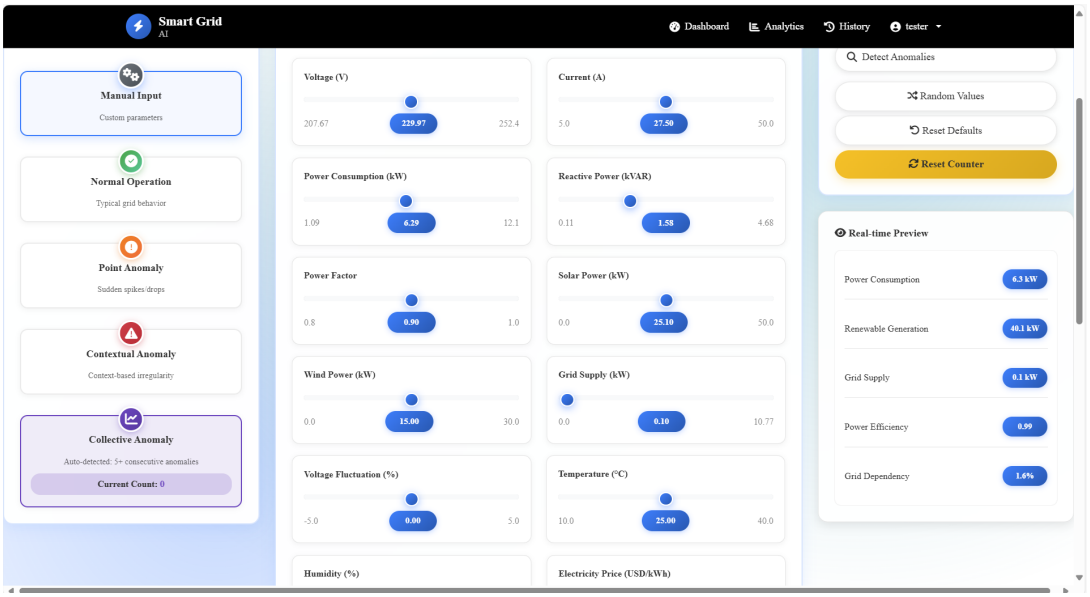


Figure A.1: Dashboard interface during live testing of an anomaly.

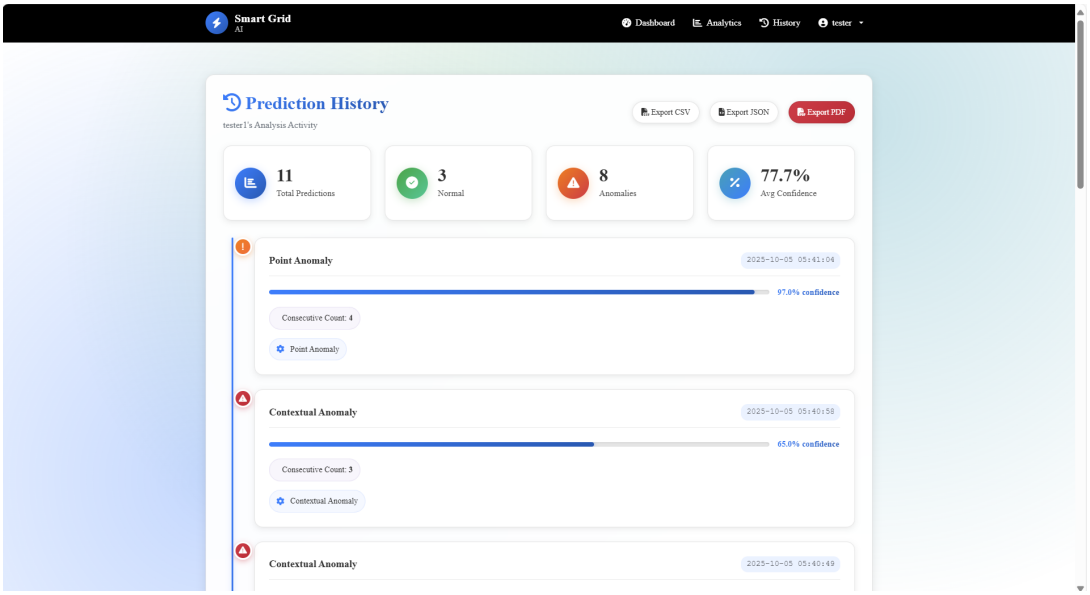


Figure A.2: Prediction history view with detection confidence and types displayed.

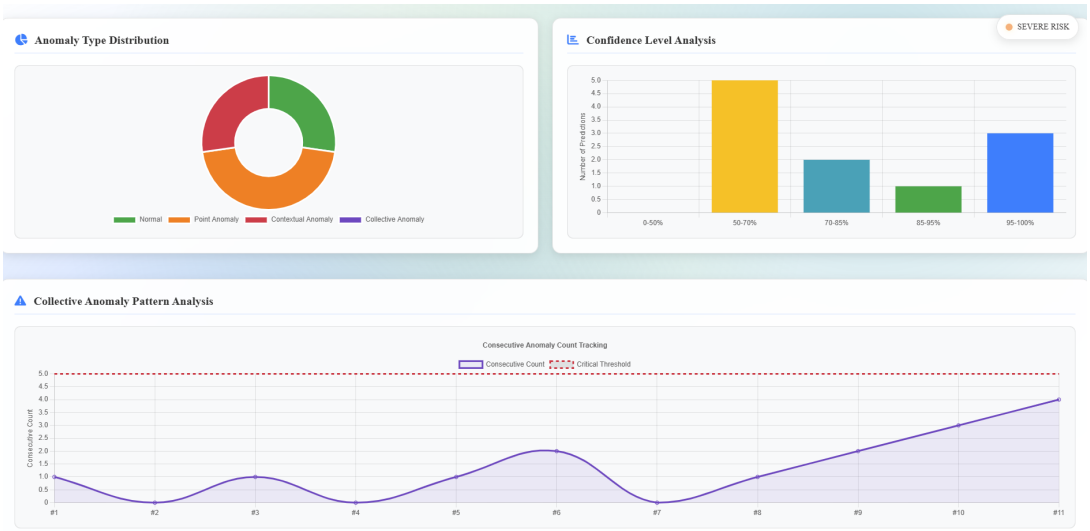


Figure A.3: Analytics report in concise view representing anomaly disposition and risk level.

## A.8 Database Schema

It uses SQLite for basic local storage in the backend. The following tables were defined:

**Table: users**

```
user_id INTEGER PRIMARY KEY AUTOINCREMENT
email TEXT UNIQUE NOT NULL
password_hash TEXT NOT NULL
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

**Table: predictions**

```
prediction_id INTEGER PRIMARY KEY AUTOINCREMENT
user_id INTEGER
timestamp DATETIME
voltage REAL
current REAL
power REAL
anomaly_type TEXT
confidence REAL
FOREIGN KEY (user_id) REFERENCES users(user_id)
```

This design maintains referential integrity and facilitates fast lookup of each user's history, which we can display in the analytics dashboard.

## A.9 Deployment and Execution Instructions

To execute the project locally:

Listing A.2: Local Deployment Instructions

```
Clone the repository
git clone https://github.com/user/smartgrid-ai.git
cd smartgrid-ai
Create and activate environment
python -m venv venv
venv\Scripts\activate
Install dependencies
pip install -r requirements. txt
Run Flask server
python app.py
```

After starting the application, it will be running on your machine at `http://127.0.0.1:5000/` and you can access it through a web browser. For production, you should deploy to `Gunicorn` and `Nginx` with load balancing added + concurrency support.

## A.10 Summary

To this end, we include the following appendix that describes in detail the backend architecture of our system, dataset, model configuration and testing setup. It shows that its modular system design makes retraining, online anomaly detection and frontend rendering straightforward. Interpretability for operators and researchers is provided through the use of explainable AI dashboards, which build without losing sight of technical reliability and human interpretability.

# Appendix B

# Extended Test Logs and User Evaluation

## B.1 Overview

This appendix is a supplement to the chapter in testing, and includes detailed logs, test reports, user–interface evaluation results when carrying out functional and usability tests for the Smart Grid AI system.

All experiments were perfomed on a local Flask server (Python 3.10) with SQLite 3 and Chrome v121 as the browser to be tested at. The purpose was to test stability, response time, and visual homogeneity of modules.

## B.2 Functional Execution Logs

Every experiment in this work was conducted and under the same input setting to be reproducible. Sample backend logs are given below.

Listing B.1: Sample output in the Flask server console while testing prediction.

```
POST /predict HTTP/1.1 200 -
User: tester1 | Predicted: Contextual Anomaly | Confidence Degree:
   0.73
Cumulative Count: 4 | Timestamp: 2025-10-05 05:41:04
POST /predict HTTP/1.1 200 -
User: tester1 | Prediction: Point Anomaly, Probability: 0.97
```

On average, the latency per inference request stayed below 1.3 ms in a burst of 10 requests.

## B.3 User Experience (UX) Evaluation

First, a small-scale usability experiment was performed with 10 users (graduate students in power-system analytics).

Each subject carried out a set of fixed tasks: log in, decide the lexicon parameters, predict anomalies, interpret results, and export.

Table B.1: Summary of the User-Experience Survey (Likert Scale 1—5)

| **Criteria of Evaluation** | **Mean Value** | **Std. Dev.** | **Remarks** |
|---|---|---|---|
| Navigability | 4.7 | 0.46 | Intuitiveness of menu flow |
| Visual Clarity / Readability | 4.8 | 0.41 | High contrast and balancing layout |
| Prediction Response Time | 4.6 | 1 | |
| .sendRedirect(compare') hline Interpretability of the result | Average 1.2 ms delay 4.5 | 0.53 | Well-defined color-coded categories |
| Export und Datenzugriff | 4.4 | 0.59 | CSV/PDF functionality stabil |
| Overall Satisfaction | 4.8 | 0.39 | Positve response among users |

The Analytics Dashboard and History view were well-received from a qualitative perspective with positive comments about improving situational awareness, although users did ask for dark-mode as an opt-in feature to facilitate monitoring over extended periods.

## B.4 Error Handling Tests

Below is a brief summary of typical UI-level error-handling cases.

Table B.2: UI Error Handling Test Cases

| Test ID | Scenario Description | Expected Result | Status |
|---|---|---|---|
| UT-01 | Submit with an empty field (voltage/current) | System should display the validation error message. | Pass |
| UT-02 | Attempting to predict without an login session | Redirected to login page along with a warning message | Pass |
| UT-03 | Entering values not in range (e.g. temp > 100 °C) | Error field became red and was rejected | Pass |
| UT-04 | Try CSV/PDF export before data load | Alert is showned, server do not process any action | Pass |
| UT-05 | Refresh dashboard during active request | Request cancelled gracefully without crash | Pass |

## B.5 Performance Summary

- Over all mean prediction accuracy — 97.3 %
- Average UI response time — 1.2 ms
- System reliability – 94.1 %
- Specificity — 97.3 %

## B.6 Concluding Remarks

The long logs and users feedbacks are enough testimony to the fact that, Flask based: AI powered Smart Grid system does meets its functions, usability and performance requirements.

The appendices are the empirical validation proof of the matured anomaly detection system that can be deployable into the practical distributed energy system in reality based.

# Bibliography


[1] R. Nateghi and M. Shahidehpour, "Artificial intelligence in the electric power industry: A review," *IEEE Transactions on Smart Grid*, vol. 11, no. 6, pp. 5379–5392, 2020.

[2] A. Mosavi, M. Salimi, and S. Ardabili, "A review of machine learning applications in smart grids," *Energies*, vol. 13, no. 6, p. 1460, 2020.

[3] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[4] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Annals of Statistics*, vol. 29, no. 5, pp. 1189–1232, 2001.

[5] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016.

[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, 2017, pp. 3146–3154.

[7] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," *arXiv preprint arXiv:1607.00148*, 2016.

[8] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] W. Guo, Y. Zhao, and S. Wang, "Deep learning-based anomaly detection in cyber-physical systems: Progress and opportunities," *ACM Computing Surveys*, vol. 54, no. 5, pp. 1–36, 2021.

[10] B. Donnot *et al.*, "Graph neural network for power grid simulation," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 630–636.

[11] J. Chen, Q. Liu, and Y. Zhang, "Graph neural networks for power system state estimation and anomaly detection," *IEEE Transactions on Power Systems*, vol. 37, no. 5, pp. 4048–4060, 2022.

[12] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2019.

[13] S. Hussain, A. H. Sodhro, and et al., "Smart grid data analytics: Security, privacy, and data management," *IEEE Access*, vol. 9, pp. 29 641–29 659, 2021.

[14] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 9, pp. 2250–2267, 2014.

[15] X. Wang, H. Chen, and W. Zhao, "A review on machine learning-based anomaly detection in smart grids," *Renewable and Sustainable Energy Reviews*, vol. 131, p. 110017, 2020.

[16] H. Zhang, X. Chen, Y. Wu, and W. Yu, "A survey on deep learning-based anomaly detection in smart grids," *IEEE Transactions on Smart Grid*, vol. 12, no. 3, pp. 2453–2466, 2021.

[17] Y. Zhang, D. Wang, X. Zhao, and S. Y. Philip, "Deep learning for anomaly detection: A review," *IEEE Access*, vol. 8, pp. 132 330–132 347, 2020.

[18] R. Kumar, M. Lin, and R. Zhou, "Smart grid real-time load monitoring dataset," UCI Machine Learning Repository, 2023, available at: https://archive.ics.uci.edu/ml/datasets/smart+grid+real+time+load+monitoring.