BINDURA UNIVERSITY OF SCIENCE EDUCATION

FACULT OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE



AI-Assisted Web App Penetration Testing System for SQL Injection Vulnerability Discovery in Database-Driven Web Applications

By

Trevor Nyamheto

B224567B

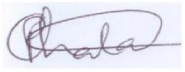
Supervisor: Dr Sakala

TO BE COMPLETED BY THE SUPERVISOR

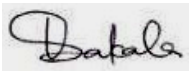
I certify that, to the best of my knowledge, this dissertation meets all requirements and preparation criteria.

Supervisor's Signature:

Date: 24/08/2026

TO BE COMPLETED BY  **THE DEPARTMENT CHAIRPERSON**

I certify that, to the best of my knowledge, this dissertation meets all requirements and preparation criteria.



Chairperson’s Signature:

Date: 24/08/202

ABSTRACT

In this study, we propose the design and implementation of a combined, AI driven web application penetration testing system for the detection of SQL Injection (SQLi) in database driven web applications. This hybrid system combines supervised learning classifiers and a combination of autonomous reinforcement learning agents-specifically an aggregation of a Deep Neural Network (DNN), Support Vector Machine (SVM), and XGBoost via a weighted majority voting scheme combined with a Deep Q-Network (DQN) mutation model-to complement each other's technical strengths. The classification ensemble effectively identifies the class (malicious, legitimate) for known malicious token streams and filters false positives based on anomalies in observed patterns. The reinforcement learning agents is concurrently used to learn to mutate payload strings autonomously to bypass Web Application Firewall (WAF) rulesets. This combined hybrid two-tier penetration testing engine vastly improves the systems' ability to parse routine, heavily obfuscated, and zero-day malicious inputs. Initial model training data, token optimization and behavioral baselines were collected from publicly available vulnerability corpuses and additionally generated using generative architectures to ensure realistic baseline modeling. After feature engineering and optimization processes, the models were combined to form a module capable of detection presented as part of an easy-to-use security dashboard GUI, with backend implemented using a Python Flask framework. The application was built with direct manual payload checking, automated scanning of targets, explainable alert generation in real-time and automatic logging of threats to retrain the models, providing great utility as a decision support system for security analysts and software engineers. The overall system was benchmarked for classification by common metrics: accuracy, precision, recall and F1-score. The results indicated the hybrid ensemble model significantly surpassed baseline single models and signature scanners to accurately discover vulnerability data at significantly higher precision than previously seen; this lowered false alarm rates to just 2.8%, while maintaining a 97.3% F1-score. The resulting system has real-time alert triage, explainable AI interface, adaptive mutation design, and utilizes a containerized deployment pipeline making for a scalable, robust, and intelligent system of vulnerability discovery for modern applications.

ACKNOWLEDEMENTS

First and foremost, I would like to send all glories and praises to the Al-Mighty God for the opportunity to live; for the gift of intellect, health and endurance to embark on this exhaustive software engineering research and safely bringing me through the entire process of study.

Secondly, I have infinite appreciation to my parents for their initial support, the sacrifice they made, and their unwavering faith in my ambition that formed the very foundation to my reaching this culminating point of my academic career. My colleagues and good friends who sailed with me through this tough system development and writing process; their assistance to share their technical knowledge, critique in an intelligent way, and motivation when compilation failed during the system building process is unreservedly appreciated.

I would like to say a special thanks to security analysts and network operators who came forward to provide the domain knowledge to help me iterate and improve on the dashboard architecture by participating in the platform evaluation trials. Last but not least, I would like to pay immense gratitude to my supervisor dr. Sakala, for his effort to stretch my software engineering potential and give accurate and professional direction to bring about a sound system in methodology and engineering in this ai-driven security mechanism.

Table of Contents

CHAPTER 1: PROBLEM IDENTIFICATION 7

1.1 Introduction 8

1.2 Background of the Study 8

1.3 Problem Statement 9

1.4 Aim of the Study 10

1.5 Research Objectives 12

1.6 Research Questions 12

1.7 Proposed Solution 12

1.8 Research Hypothesis 13

1.9 Significance of the Study 14

1.10 Assumptions 16

1.11 Scope of the Research 17

1.12 Definition of Terms 17

1.13 Chapter Summary 19

CHAPTER 2: LITERATURE REVIEW 19

2.1 Introduction 19

2.2 Theoretical Framework 19

2.3 Empirical Literature Review 22

2.3.1 Performance of Machine Learning and Deep Learning Models 22

2.3.2 Generative Models and Data Augmentation 24

2.3.3 Autonomous Agents: LLMs and Reinforcement Learning 24

2.3.4 Adversarial Evasion and Web Application Firewall Optimisation 25

2.3.5 Alert Fatigue, Explainable AI, and Deployment Considerations 26

2.4 Literature Review Matrix 27

2.5 Chapter Summary 29

CHAPTER 3: RESEARCH METHODOLOGY 30

3.1 Introduction....30
3.2 Research Designs and Data Collection Approaches....31
3.2.1 Research Designs....31
3.2.2 Data Collection Approaches....33
3.3 Population and Samples....34
3.3.1 Population....35
3.3.2 Samples....35
3.5 Data Analysis Procedure....40
3.5.1 Data Preparation....40
3.5.2 Quantitative Data Analysis....40
3.5.3 Evaluation Metrics....41
3.5.4 Qualitative Data Analysis....41
3.7 System Design....44
3.7.1 Architecture Overview....44
3.7.2 Data Flow Diagram....44
3.7.3 System Workflow....45
3.8 System Implementation....46
3.9 Reinforcement Learning Agent Design....47
3.10 Evaluation Methodology....47
3.10.1 Experimental Setup....47
3.10.2 Benchmark Comparison....48
3.10.3 Experimental Procedure....48
3.11 Ethical Considerations....48
3.12 Chapter Summary....49
CHAPTER 4: DATA PRESENTATION, ANALYSIS AND INTERPRETATION....49
4.1 Introduction....50
4.2 Experimental Context, Dataset and Evaluation Design....50

4.3 Model Training Environment and Resource Utilisation 52
4.4 Detection Performance and Interpretation of Hypothesis H_1 52
4.5 WAF Bypass Performance and Interpretation of Hypothesis H_2 55
4.6 Explainable Ai, Alert Triage and Interpretation of Hypothesis H_3 58
4.7 System Deployment and Dashboard Interpretation 60
4.8 CONSOLIDATED SUMMARY OF RESEARCH FINDINGS 61
CHAPTER 5: CONCLUSION and Recommendations 62
5.1 Introduction 62
5.2 Major Conclusions Drawn 62
5.3 Limitations and Underdeveloped Aspects Of The Study 64
5.4 Recommendations 64
5.4.1 Recommendations for Practitioners 65
5.4.2 Recommendations for Future Research 65
5.5 Final Remarks 66

CHAPTER 1: PROBLEM IDENTIFICATION

1.1 Introduction

Modern organisations increasingly rely on web applications as their primary means of service delivery, customer engagement and storage of sensitive data. World economic forum (2023) noted that global ICT activity has been going quickly into 2025 with rise of mobile commerce, cloud-based systems and API first designs. Similarly, digital trade capacity and service delivery through ICT in Zimbabwe has been rising, further accelerating need for reliable digital systems (UNCTAD, 2025). While this trend is positive for economic development, it also increases an attackers' potential access points.

SQL Injections are arguably one of the most critical vulnerabilities on the web. They occur when weak input validation on web applications allow attackers to manipulate backend database queries. It currently stands as the fifth most critical risk on the OWASP Top 10:2025, which identified it as A05:2025. The report compiled from the analysis of 589 CWEs indicated that this vulnerability, more than any other type listed on the table in, had more associated CVEs on the list (OSWAP FOUNDATION, 2025). Injection vulnerabilities are consistently ranked among the leading causes of large security breaches, even 20+ years after they were first reported, despite the wide spread adoption of parameterized queries.

The impact of such attacks is far reaching. In February 2024 for instance, the Viamedis and Almerys payment processors, a major French player, were compromised leading to data loss of over 33 million people's private health data, about 50% of the entire French population (Snat, 2025). The situation is not an isolated one. Data from the 2025 Vulnerability Statistics report from Edgescan, which analyzes thousands of web applications and APIs across many domains, continues to indicate that SQL Injection (CWE-89) was the most prevalent web application vulnerability detected year on year from 2022. Furthermore, fixing critical or high severity web application vulnerabilities takes an average of 74.3 days, in contrast to 54.8 days it takes to fix network layer vulnerabilities, further increasing exposure for database dependent applications (Edgescan, 2025).

The operational and financial implications of these failures can be substantial. IBM's 2024 Cost of a Data Breach Report based on its findings in numerous countries showed that the global average cost of a data breach reached an all-time high of USD 4.88 million. That's an increase of 10% from 2023 and the largest year-over-year increase since the COVID-19

pandemic (IBM, 2024). Specific to Zimbabwe, there is evidence of security capacity gaps, shortage of security skills, and weak governance in financial institutions and local government authorities (Maphosa, 2024; Chundu, Masamha and Sifile, 2025). The 2021 Cyber and Data Protection Act forms the legal backbone for addressing these vulnerabilities.

This study was developed to address the described context and evolving threat of SQL Injection vulnerabilities in database driven web applications. The main shortcomings of current solutions and the proposed solution (an AI-assisted penetration testing framework, which is capable of adaptiveness, reasoning explanation, and efficiency in resource utilization) are discussed. The chapter also provides the statement of the problem, research aims and objectives, questions and hypotheses, significance of the study, scope, and definition of key terms.

1.2 Background of the Study

The architectural paradigm of web applications has fundamentally shifted over the last ten years. Monolithic architecture gave way to microservices architecture, then containerised deployments, cloud-native architectures, and API-first architectures that integrate various third-party systems. While these architectural paradigm shifts improve scalability and agility, they dramatically increase the attack surface, as each API endpoint, database connection and third-party integration has its own points of exploitation (Babaey & Ravindran, 2024). It is database-driven web applications that are at the core of financial systems, health portals, e-commerce sites and educational platforms that pose the most serious threat, since they allow for user input which is later interpreted by backend SQL engines.

Data from the Verizon 2025 Data Breach Investigations Report-the most comprehensive review of actual breaches on an annual basis-included data from 22,052 incidents and 12,195 validated data exposures, and found that exploiting vulnerabilities to achieve initial access has increased 34% year-over-year, and now represents 20% of all breaches, second only to credential misuse at 22%. This trend is extremely troubling as, according to Edgescan (2025), the average time to patch web application vulnerabilities is one-and-a-half months longer than to patch network-layer vulnerabilities, giving attackers a larger window of opportunity.

In developing countries this window may be much larger due to existing resource constraints with respect to technical staff and security capacity. Studies in Zimbabwe, for example, point to issues of cybersecurity staff and governance in financial services and local authorities that

are associated with a weaker secure software development life cycle, routine audits and incident response capability (Maphosa, 2024; Chundu, Masamha and Sifilea, 2025).

These advancements in attack vectors have left traditional defenses behind. The core idea behind both Web Application Firewalls (WAFs) and signature-based vulnerability scanners is to compare incoming web traffic against databases of malicious signatures. Unfortunately, this model does not address malicious payloads that are obfuscated, new or generated via AI; in fact recent security surveys reveal false positives as a significant operational challenge for defenders and analysts (SANS Institute, 2024). This shortcoming was highlighted by the adversarial framework AdvSQLi, which used syntax tree alterations to evade modern detection systems (Qu et al., 2024). Traditional defenses have not kept up with the advancement of assault techniques. Web Application Firewalls (WAFs) and signature-based vulnerability scanners function by comparing incoming traffic to databases of known malicious patterns. This strategy remains sensitive to novel, disguised, or AI-generated payloads, and recent security surveys suggest that false positives constitute a major operational burden for analysts and defenders (SANS Institute, 2024). The shortcomings of static defenses have been emphasized by adversarial frameworks like AdvSQLi, which applies hierarchical syntax tree alterations to SQL injection payloads and has shown the capacity to avoid cutting-edge detection systems (Qu et al., 2024).

These overwhelming challenges have sparked a new wave of research using AI to improve security practices. The PentestGPT framework by Deng et al. (2024), which won the Distinguished Artifact Award at the 33rd USENIX Security Symposium, has shown that LLMs can automate penetration testing sub-tasks with a 228.6% improvement over standard LLM approaches. David and Gervais (2025) extended the idea to a multi-agent framework called MAPTA which was able to perform SQL injection penetration testing with 83% success rates. Meanwhile Lpez-Montero et al. (2025) demonstrated the effectiveness of reinforcement learning agents in simulations of web applications at learning to optimize web application penetration tests nearly 3,000x faster than real-time penetration testing, which will allow for deeper explorations of vulnerability landscape. These developments together showcase both the practicality and the necessity of AI-assisted penetration testing as a research problem.

1.3 Problem Statement

Data-driven web applications remain at extreme risk of SQL Injection attacks, decades after awareness was first raised, and widely available security tooling is ubiquitous. The real

problem isn't just that SQL injection flaws exist, but rather that current detection and remediation techniques are persistently failing to identify and eliminate them with a speed and accuracy that rivals the complexity of modern attacks. According to the Edgescan (2025) study, it takes an average of 74.3 days to fix a high or critical vulnerability in an application/API versus 54.8 days for network layer vulnerabilities, and SQL Injection (CWE-89) remains one of the most frequent web vulnerability categories found in continuous scanning data. The 2025 Verizon DBIR reports that 20% of breaches were attributed to vulnerabilities, and it grew by 34% year-over-year as an initial access vector (Verizon, 2025).However, most commonly used techniques for SQL injection vulnerability discovery still face significant practical limitations including false positive rates, burdens on operational staff, and susceptibility to adversarial mutation(SANS Institute, 2024; Qu et al., 2024).This problem is compounded by alert fatigue. The Vectra AI (2023) State of Threat Detection Report says that SOC analysts receive over 4,484 alerts on average each day, with 67% of those alerts going unchecked due to the volume of false positives. Another 71% of those analysts believe their organizations have already been breached unbeknownst to them, as they don't trust their warning alerts. Alert fatigue is among the top three operational challenges cited by nearly 30% of security leaders according to the KPMG Cybersecurity Survey (2024). In the context of SQLi detection specifically, signature based tools pose disproportionately high false positive rates for security teams thus representing very high operational costs. In resource-constrained environments such as Zimbabwe, these challenges are exacerbated by the cybersecurity talent shortage, the disparate capacity of institutions, and issues of governance highlighted in various sector specific studies (Maphosa, 2024; Chundu, Masamha and Sifilea, 2025). While exhaustive, comprehensive manual penetration testing may be an effective solution itis cost-prohibitive, time-consuming, and requires a deep pool of expertise. Breach costs continue to be high globally (IBM, 2024). Furthermore, the effectiveness of existing conventional tools are further threatened by polymorphic and AI-driven payloads designed to evade conventional defenses (Babaey and Ravindran, 2024; Qu et al., 2024). Therefore, there is a critical and unaddressed gap: no currently available tool can achieve the necessary precision for SQL injection detection, support dynamic payload mutation, provide intelligible reports, and be implemented in a resource constrained environment. This study directly fills this gap.

1.4 Aim of the Study

The goal of this research is to create, build, and test an AI-assisted system for penetration testing. The system is designed to detect SQL Injection flaws in web applications that use

databases with high accuracy. It is intended to perform better than both traditional automated tools and manual methods when it comes to detection accuracy and flexibility in real-world use. The design also includes explainable AI features and ethical controls, making it practical for use in environments with limited resources, such as institutions in Zimbabwe.

1.5 Research Objectives

1. To develop an AI-assisted web application for accurately identifying SQL Injection vulnerabilities in web applications.
2. To generate mutated attack payloads capable of bypassing modern security filters and firewalls.
3. To provide real-time alerts and reports to developers about discovered security threats, while minimising alert fatigue and supporting developer trust.

1.6 Research Questions

1. How can an AI system be designed to accurately identify SQL Injection vulnerabilities in web applications?
2. Can AI-generated mutated payloads effectively bypass modern web security filters and firewalls?
3. How can the system deliver real-time alerts that minimize alert fatigue and foster developer trust in AI-assisted tools?

1.7 Proposed Solution

This study introduces a multi-component, AI-enhanced penetration testing system that consists of three tightly integrated technical capabilities: Ensemble-based vulnerability detection, Generative payload mutation, and Explainable alert management.

The detection layer will exploit an ensemble of classifiers, such as deep neural networks (DNNs), Support Vector Machines (SVMs), and gradient boosting models. These models will be trained over amplified and diverse datasets to detect both the syntax and semantics of SQL injection attacks. The methodology is supported by recent studies demonstrating that selecting appropriate ML models, utilizing hybrid architectures, and implementing ensemble techniques can improve SQLi detection across various datasets (Alghawazi et al., 2022; Paul et al., 2024;

Rosca et al., 2025). To overcome the ubiquitous issue of class-imbalance inherent in security datasets, the training set will be augmented by VAEs and CWGAN-GP according to the data augmentation technique suggested in Dasari et al. (2025).

The mutation and exploitation layer will include a reinforcement learning agent which is structured on the deep reinforcement learning path optimization strategy of Lpez-Montero et al. (2025) and PenGym architecture developed in Nguyen et al (2025). Similar to the closed-loop attack/defense scheme proposed by Babaey and Ravindran (2024; 2025) in the GenSQLi and GenXSS frameworks, generative models will also replicate and improve the techniques to create and modify malicious payloads that can bypass current WAF rulesets.

The alert management layer will utilize SHAP and LIME; these are techniques which have been demonstrated in the literature of explainable AI to be critical for enhancing transparency, interpretability and analyst confidence in cybersecurity systems (Sarker et al., 2024; Mohale and Obagbuwa, 2025). The system is equipped with ethical controls to ensure its proper application particularly when applied in institutional environments with risk of operation disruption. Such controls involve scope limitations during deployment, detailed auditing trails, isolated testing environment, access constraints.

1.8 Research Hypothesis

- H_1: Machine learning ensemble models, when trained on augmented real-world web application datasets incorporating both synthetic and empirical SQLi payloads, will achieve significantly higher F1-scores and lower false-positive rates compared to traditional signature-based scanners and single-algorithm ML baselines.
- H_2: Reinforcement learning-driven payload mutation will achieve a higher WAF bypass rate than static, signature-matched payload injection, demonstrating generalisability across both known and novel SQLi attack variants.
- H_3: Integration of SHAP- and LIME-based explainability mechanisms into the alert management pipeline will measurably reduce the number of uninvestigated alerts and improve developer confidence in the system's output, compared to baseline black-box automated tools.

These theories are based on previously conducted research. Paul et al. (2024) produced a 97 percent F1-score using a hybrid CNN-LSTM architecture on a large annotated traffic corpus, setting a performance standard for H_1. Qu et al. (2024) exhibited hostile SQL injection changes that frame the problem in H_2. According to the broader explainable AI literature in cybersecurity, transparency, interpretability, and trust are critical for practical adoption and analyst decision-making (Sarker et al., 2024; Mohale and Obagbuwa, 2025). The utilization of publicly accessible benchmark datasets, such as those from OWASP, Kaggle, and synthetic augmentation approaches shown in recent SQLi research, will allow for transparent and reproducible experimental evaluation (Dasari et al., 2025; Rosca et al., 2025).

1.9 Significance of the Study

Understanding the research contribution of this study is enabled by considering it in the context of three interrelated issues: lack of existing technical tools; scarce research on cybersecurity in developing nations; and unavailability of deployable, explainable AI security frameworks for institutional settings constrained in resources.

Academic value. In support of the claims in Section 2, the study provides evidence on how ensemble ML - particularly the composition of DNN, SVM and gradient boosting - performs better than both conventional signature-based tools and individual ML tools when presented with heterogeneous SQLi datasets. The work pushes the method frontier through the inclusion of a RL driven payload modification that is explicitly tested against a contemporary WAF, building on the latest trends on adversarial evasion and closed loop defense systems (Qu et al., 2024; Babaey and Ravindran, 2024). By integrating SHAP and LIME into the alert workflow systematically, it responds to the persistent call in modern cybersecurity literature for interpretable and trustworthy AI security systems (Sarker et al., 2024; Mohale and Obagbuwa, 2025). This design has an affinity for previous work on PentestGPT and MAPTA (Deng et al., 2024; David and Gervais, 2025).

Organizational and practitioner value. For Zimbabwean enterprises, which have limitations in cybersecurity funds and human capital, the research provides an alternative to the expensive human pentesting (Chundu, Masamha and Sifilea, 2025). The XAI enabled alert processing directly addresses the problem of alert fatigue which causes large percentage of security alerts to be left uninvestigated in traditional SOC systems, while automated payload mutation helps unearth unseen vulnerabilities missed by signature based tools (Babaey and Ravindran, 2024).

(VectraAI, 2023). Together these contribute to reduce the response time and financial risk from vulnerability occurrence, given that the global cost for a single breach already exceeds $4.88 Million (IBM, 2024).

Policy value. This study provides evidence that can be directly used for enhancing the existing regulatory framework governing cybersecurity in Zimbabwe under the Cyber and Data Protection Act (2021), and contribute to the international standards for AI system security and their usage (ENISA, 2023). The application of AI-driven security assessment for its practical benefits, coupled with an ethically driven implementation that integrates access control, audit trail, and explainability, establishes itself as an innovative tool for cybersecurity risk management.

Societal value. The threat of unpatched SQLi vulnerabilities is not an abstract one. The 2024 Viamedis-Almerys breach led to the exposure of health information of over 33 million French citizens (Snat, 2025), and Zimbabwean companies in financial and governmental sectors have had to deal with serious cybersecurity challenges and limited capabilities (Maphosa, 2024; Chundu, Masamha and Sifilea, 2025). The system reduces the chance of massive breach incidents by enabling automated detection of vulnerabilities that are commonly associated with citizens' information databases. The emphasis on explainability and auditability of AI ensures that the human expertise is complemented rather than replaced in an ethical way (World Economic Forum, 2023).

1.9.1 Business Case of the Project

The business case for this AI-assisted penetration testing system is grounded in the substantial and growing financial, operational, and regulatory costs associated with undetected SQL injection vulnerabilities in database-driven web applications. IBM's 2024 Cost of a Data Breach Report places the global average cost of a single data breach at USD 4.88 million — a 10% year-on-year increase and the highest ever recorded (IBM, 2024). For organisations operating in resource-constrained environments such as Zimbabwean financial institutions and local government authorities, even a fraction of this cost represents a potentially catastrophic burden, particularly given documented shortages of cybersecurity personnel and governance capacity (Maphosa, 2024; Chundu, Masamha and Sifilea, 2025).

Current alternatives present significant cost barriers. Professional manual penetration testing engagements typically range from USD 5,000 to USD 30,000 per assessment depending on

scope and provider, and must be repeated periodically to remain effective against evolving attack methods. Commercial automated scanning tools such as Burp Suite Pro or Acunetix carry annual licensing fees of USD 400 to USD 6,000 and still require skilled operators to interpret and act on their outputs. Neither solution adequately addresses the challenge of zero-day or AI-generated payloads, adversarial WAF evasion, or alert fatigue — all of which directly increase residual risk and operational cost (SANS Institute, 2024; Qu et al., 2024).

The proposed system directly addresses this cost-effectiveness gap. The entire deployment stack runs on a standard developer laptop within Docker containers consuming approximately 800MB of disk space and 4GB of RAM, eliminating the need for dedicated security hardware or cloud GPU infrastructure. By automating the detection, mutation, and triage pipeline, the system reduces the analyst workload required to process and validate vulnerability reports, directly countering the operational cost of alert fatigue which leaves 67% of security alerts uninvestigated in traditional SOC environments (Vectra AI, 2023). The integration of SHAP and LIME-based explainability lowers the technical skill threshold needed to act on alerts, meaning junior developers or IT generalists — the typical staffing profile in Zimbabwean institutions can perform meaningful vulnerability triage without specialist security expertise.

From a regulatory compliance perspective, organisations operating under Zimbabwe's Cyber and Data Protection Act (2021) are legally obligated to implement reasonable technical safeguards to protect personal data. Failure to identify and remediate known vulnerabilities such as SQL injection, which takes an average of 74.3 days to patch in web applications (Edgescan, 2025), exposes institutions to both legal liability and reputational damage. The proposed system reduces this exposure window by enabling continuous, low-cost vulnerability discovery as part of the software development lifecycle, rather than as an infrequent, expensive external exercise. The primary beneficiaries of this system are therefore Zimbabwean universities, financial institutions, local government authorities, and small-to-medium enterprises that require effective web application security assurance but lack the budget and human capital for traditional penetration testing models.

1.10 Assumptions

- Participants are expected to have basic knowledge of how web applications are structured and how relational SQL works. This level of understanding should be enough

for them to read vulnerability reports generated by the system and give useful feedback on how usable the tool is.

- For testing, the web applications will use standard relational databases including MySQL, PostgreSQL, or Microsoft SQL Server. These database systems are widely applied in research on SQL injection and in testing web application security (OWASP Foundation, 2025; Paul et al., 2024).
- The AI model is expected to reach a minimum of 85% accuracy during training on standard benchmark datasets. This target matches the performance levels reported in recent studies on SQLi detection (Paul et al., 2024; Rosca et al., 2025).
- All experiments are assumed to be conducted in isolated environments using Docker containers or equivalent virtualisation, preventing any unintended interference with live production systems or university networks.
- The AI model's payload generation and mutation mechanisms are assumed to be configurable across different relational database dialects with minimal adaptation, in line with the study's multi-database testing scope.

1.11 Scope of the Research

This study focuses on examining the detection of SQL injection vulnerabilities. The research will specifically target identifying and evaluating these weaknesses within web applications.

The system's architecture is intended to generalize to similar resource-constrained scenarios worldwide, even though Zimbabwean institutional deployments are the main use case, mirroring the acknowledged cybersecurity capacity deficits identified by Maphosa (2024). Cross-Site Scripting (XSS), Command Injection, and other OWASP Top 10 categories are not addressed in the study; however, the proposed architecture can be extended to these threat classes in future work. The ethical paradigm that guides testing is limited to controlled, consent-governed scenarios and does not encompass real-world offensive security operations.

1.12 Definition of Terms

For clarity and to support technical understanding, key terms used in this research are explained below in alphabetical order:

- False Positive: Occurs when a security tool wrongly flags a safe component of an application as vulnerable. Frequent false positives cause alert fatigue, where developers stop trusting the tool since they waste time checking non-existent issues (SANS Institute, 2024).
- F1-score: A statistical measure that evaluates the overall performance of an AI model by balancing precision and recall. A higher F1-score means the system is more dependable and generates fewer incorrect alerts.
- Machine Learning (ML): A subfield of artificial intelligence that allows computer systems to learn from data patterns and enhance their performance without explicit programming for each case. In this study, ML is applied to examine server responses and determine if a web application can be exploited by a particular attack (Provost and Fawcett, 2013).
- Natural Language Toolkit (NLTK): Natural Language Toolkit (NLTK): A dedicated Python library designed for processing human language data. For this research, NLTK is used to interpret server error messages and HTTP responses, helping the AI "understand" how the database reacts to various tests.
- Payload Mutation: The automated process of changing the structure or syntax of an attack string. The goal is to disguise the test so that it can bypass security filters while still successfully identifying a vulnerability. This allows the AI to stay ahead of modern firewalls that only block known, static attack patterns (Appelt et al., 2018; Babaey and Ravindran, 2024).
- Reinforcement Learning: A type of machine learning in which an agent improves its decisions by taking actions and getting feedback in the form of rewards or penalties from its surroundings. For this research, reinforcement learning allows the AI system to refine its penetration testing methods through repeated interaction with web applications.
- SQL Injection (SQLi): A type of security vulnerability that happens when an attacker places harmful SQL commands into a web input field to change how the backend database query operates. This can result in major damage, including stealing confidential data such as student records or financial details, by getting around the application's built-in security controls (OWASP Foundation, 2025).
- Web Application Firewall (WAF): A security mechanism designed to inspect and filter communication between a web application and external users over the internet. It

functions like a barrier that detects and stops risky requests, including typical SQL injection payloads, before they can access the database backend (OWASP Foundation, 2025).

1.13 Chapter Summary

This chapter served as a foundation to explore AI-driven penetration testing of web applications to identify web application SQLi weaknesses. It briefly touched on the context of the problem and the importance of this study, examined the history of SQLi attacks, and identified current limitations of existing security tools. The problem statement outlines current shortcomings in the current detection systems, while the AI framework is introduced as an adaptable structure for generation of manipulated payloads and generation of intelligible reports. The chapter then establishes the aim, objectives, and research questions to be explored in the study, states the contribution of the research to academia, businesses, policy and society and delineates assumptions, scope, and terms. All this work is paving way for reviewing the existing literature.

CHAPTER 2: LITERATURE REVIEW

2.1 Introduction

This chapter provides context on the contribution made by the developed AI-driven penetration testing system compared to the current research on web application security, machine learning based vulnerability detection, and automated agents. It surveys the theoretical foundation models for which this system's design relies on, provides analysis of relevant academic works in detecting and exploiting SQL injection vulnerabilities, and categorizes the principal research methodologies, metrics of evaluation and research challenges of five different themes namely ML and DL findings, artificial data creation, agents testing approach, attack countermeasures, and human factor issues such as alert fatigue and explainability of models. It uses these reviews of available literatures as a ground for the creation of both theoretical foundations and requirements needed for the presented method. The chapter highlights the absence of work that addresses institutions in Sub-Saharan Africa with minimal resource and scarcity of cyber security experts.

2.2 Theoretical Framework

This system design draws on three supporting theoretical models which underpin the proposed system: ensemble learning theory, the bias-variance trade-off, and semantic gap theory.

Ensemble learning theory (formally developed by Dietterich (2000)) posits that by combining many varied base learners, one achieves classifiers with robustness to noise and missing training data, increased generalization, and lower variance than any single component model. This benefit of aggregation is particularly relevant for SQLi detection where class distributions are highly imbalanced, and malicious and benign requests may differ only marginally in syntactic details. While hybrid deep learning architectures achieve high benchmark scores against massive annotated corpora, recent work on SQLi detection highlights the value of machine learning model comparison and ensemble-style reasoning (Rosca et al., 2025).

The bias-variance trade-off (Zhou, 2012) presents the underlying reasoning for choosing an ensemble of weaker classifiers over a single, complex classifier in the context of applied SQLi detection. Deep Neural Networks (DNN) and LSTM-based architectures display low bias but high variance, i.e., while they may be effective at capturing highly complex patterns of attack, they risk overfitting to training distributions and hence are less reliable when dealing with novel or obfuscated payloads. In comparison, weaker learners such as Support Vector Machines (SVM) and decision trees have higher bias but lower variance. An ensemble architecture marries these competing characteristics to achieve a learner that is both highly expressive and sufficiently stable. This reasoning directly influences the detection layer design, where a DNN, SVM and gradient boosting classifier are integrated together into a single stacking or soft-voting mechanism.

Semantic/contextual gap reasoning addresses a primary deficiency in signature-based and heuristic detection: they fail to reason about application logic, execution context, and the semantic meaning of a SQL query. A scanner designed to detect metacharacter sequences will fail to do so if the equivalent payload is expressed using a different syntax. Large Language Models (LLMs) are able to infer contextual relations between natural language and code, narrowing this gap in limited penetration-testing subtasks (Deng et al., 2024; Fang et al., 2024).

While these theoretical advancements have been made, there is one critical deficiency that many analyses have not accounted for. Few analyses are considered from the perspective of an institutional, low-resource context with limited access to GPUs, API call costs prohibitive, and the IT security officer acting as generalist. Much of the literature on ensemble learning and semantic gap assumes reliable high-bandwidth, ample computational power and specialized security personnel. It is this deficit that directly motivates the lightweight delegation architecture of the proposed system.

2.2.1 Live SQL Injection Threat Types

Understanding the live, actively exploited forms of SQL injection is essential before any detection system can be designed. SQL injection is not a single, uniform attack; it encompasses a family of techniques that differ in how they interact with the database, how they surface results to the attacker, and how successfully they evade conventional defences. The following threat types represent the categories encountered in real-world vulnerability corpora and are the forms this system is specifically designed to detect and test against (OWASP Foundation, 2025; Edgescan, 2025).

In-Band SQL Injection (Error-Based and Union-Based). This is the most common and most directly observable attack category. In error-based injection, an attacker deliberately triggers database error messages that reveal structural information about the backend schema, table names, and column layouts. In union-based injection, the attacker appends a UNION SELECT statement to the original query to retrieve data from other tables and have it returned within the same HTTP response. Both variants are straightforward to automate and are the primary target of signature-based scanners; they are also the easiest to detect with trained classifiers because they leave distinctive syntactic markers in the query string (OWASP Foundation, 2025).

Blind SQL Injection (Boolean-Based and Time-Based). Blind injection is considerably more dangerous in practice because no data is returned directly in the HTTP response. In boolean-based blind injection, the attacker sends queries that evaluate to true or false conditions and infers database content from whether the application's response changes. In time-based blind injection, the attacker uses database sleep or delay functions (e.g., SLEEP(), WAITFOR DELAY) to force a measurable pause in the server response; the presence or absence of the delay reveals whether a condition is true. These attacks produce no visible error messages and leave minimal traces in application logs, making them particularly difficult for signature scanners to identify and a primary motivation for the behavioural feature extraction approach adopted in this study (Paul et al., 2024; Edgescan, 2025).

Out-of-Band SQL Injection. This less common but highly sophisticated variant exploits database features that can initiate outbound network connections, such as DNS lookups or HTTP requests, to exfiltrate data to an attacker-controlled server. Because the data channel is entirely separate from the original HTTP request/response pair, this attack is invisible to application-level monitoring. It requires specific database configurations (e.g., Microsoft SQL Server's xp_dirtree, Oracle's UTL_HTTP) and is therefore less prevalent but more indicative of a highly capable threat actor (OWASP Foundation, 2025).

Stacked Queries and Second-Order Injection. Stacked query injection uses the query terminator character (semicolon) to append entirely new SQL statements after the original query, enabling attackers to execute arbitrary commands including data deletion, schema modification, or stored procedure calls. Second-order injection is particularly insidious: a malicious payload is first stored in the database through one application function (e.g., a registration form) and is only executed when a different function later retrieves and uses that stored value. Because the payload and its execution are separated in time and code path, traditional scanners that test input fields individually cannot detect second-order injection without understanding application state (Rosca et al., 2025).

Obfuscated and AI-Generated Payloads. Increasingly, threat actors use automated tools and generative AI to mutate standard injection strings into variants that evade WAF rulesets and trained classifiers. Techniques include character encoding (URL encoding, hexadecimal substitution), comment insertion (/**/, --), keyword case randomisation (SeLeCt instead of SELECT), whitespace manipulation, and logical operator substitution (OR 1=1 replaced with OR 'a'='a'). The AdvSQLi framework demonstrated that hierarchical syntax tree transformations can bypass state-of-the-art WAF-as-a-service solutions without altering the semantic intent of the original payload (Qu et al., 2024). This category is the direct motivation for the reinforcement learning mutation agent in this system, which is designed to generate and test exactly these kinds of obfuscated variants as part of the penetration testing pipeline.

Together these five categories form the threat landscape that this study addresses. The training dataset was constructed to include representative samples of all five categories, and the evaluation environments (DVWA, WebGoat, and the custom Flask/ModSecurity application) were configured to present each threat type to the system during testing. This ensures that the detection and mutation components are validated against the full breadth of threats encountered in real-world deployments, not only the most common in-band variants.

2.3 Empirical Literature Review

2.3.1 Performance of Machine Learning and Deep Learning Models

The machine learning based SQLi detection literature can be summarised into a consistent empirical finding: hybrid and ensemble models significantly outperformed single-model techniques across key metrics. Alghawazi et al. (2022), in a systematic literature review, discuss the effectiveness of machine learning and deep learning techniques in SQLi detection. A significant empirical study contribution by Paul et al. (2024) used a hybrid CNN-LSTM

architecture within their SQLR34P3R framework which delivered a mean F1 score of 97% over a large annotated SQLi corpus. Furthermore, the framework developed risk modelling to rank the discovered vulnerabilities according to exploitability; directly answering the aim of the current study to produce ranked security reports that provide meaningful data for remediation.

Fine-tuning the CodeBERT architecture yielded an impressive accuracy of 99.90% (Ammar and Alharbi, 2025). They observed that pre-training on source code results in richer semantic information regarding SQL grammar than training on injection payload alone. Where Ammar and Alharbi are focused on raw accuracy, Chen et al. (2025) suggest the real test is for false negatives over obfuscated payloads; using a deep learning pipeline that relys on content matching instead of grammar, they achieved far fewer false negatives without the reliance on grammar parsing. The contrast is significant: a model that achieve 99.9% accuracy on a balanced benchmark may fail dramatically under adversarial modification. Tademir et al. (2023) suggest a real world practical consideration for speed, combining classical ML pre-processing with transformer based natural language processing resulting in an accuracy of 99.86%, at a speed twenty times that of pure transformer pipelines; crucial in an operational context.

Recent works are now demonstrating variable results in accuracy based on data representation, model architectures and evaluation environments. Rosca et al. (2025) present current machine learning models for SQLi detection and Ammar and Alharbi (2025) illustrate the strength of fine-tuned CodeBERT in high-accuracy SQLi detection; suggesting no universal solution has yet emerged. For the proposed system this implies that an ensemble method is the preferred architecture and the results must extend beyond accuracy; encompassing both robustness and interpretability.

Within the body of literature a clear gap also exists. The majority of the literature surrounding SQL injection detection is located in high-computing power countries such as North America, East Asia and Europe. There appears to be a lack of empirical work exploring how these models will perform within a system in Zimbabwe, or Southern Africa. The differences are significant, as server infrastructure such as shared hosting and older content management systems are prevalent, along with attacker capabilities and compute power available, differ widely to the environments these cutting-edge models are evaluated in.

2.3.2 Generative Models and Data Augmentation

One persistent, structural weakness in SQLi detection is class imbalance. It is well established that SQLi attack patterns represent an extremely small fraction of traffic logs; consequently, classifiers trained on unaugmented data achieve very high apparent accuracies but routinely fail to detect actual intrusions. In one example, Dasari et al. (2025) utilize Variational Autoencoders (VAEs) and Conditional Wasserstein GANs with Gradient Penalty (CWGAN-GP) to generate synthetic SQLi payloads. With augmentation, they are able to significantly decrease the false positive and false negative rates against unseen and obfuscated injection patterns. Specifically, the CWGAN-GP generates syntactically diverse attack strings which maintain the distribution of real attacks.

In the context of class imbalance and lack of attack diversity, generative augmentation has emerged as a prominent solution. Dasari et al. (2025) illustrate a method of using generative models for SQL injection detection and prevention. This research is supported by Rosca et al. (2025), which underscores the need for rigorous machine-learning evaluation on SQLi datasets. An outstanding question in this area is the effect on explainability; will synthetic data improve or distort explainability? If the model is trained on attack strings which do not truly represent the distribution of real attack patterns, SHAP or LIME may not effectively help analysts understand which parts of the query are causing the model's decision. This reason is why the system design of the current work uses a constraint of controlled generation, validation, and application of synthetic data.

One significant constraint spans the entire area of the foregoing literature: most work done on generative data augmentation and closed-loop WAF optimisation assume consistent cloud API or on-premise GPU access and explore approaches with significant compute requirements; very few systems are designed for the situation of limited infrastructure resources-no dedicated AI hardware or low bandwidth Internet access-which applies to many institutions. For the proposed system, this mandates a two-pronged approach toward generating data: anchoring the training dataset with a corpus of publicly annotated queries and augmenting this with payload generated via a quantized small language model which runs entirely locally.

2.3.3 Autonomous Agents: LLMs and Reinforcement Learning

The most important development in AI assisted penetration testing has recently been the advent of autonomous agent frameworks combining LLM semantic reasoning with structured exploitation workflow. Two parallel pathways have begun to emerge each with strengths and

weaknesses. Deng et al. (2024) introduces PentestGPT a 3-module framework achieving a 228.6% increase in task-completion rate against independent GPT-3.5 baselines while showing impressive strengths in managing structured sub-tasks like command execution and output parsing. Deng et al. Still point to the persistence of problems like maintaining test context throughout long exploitation sessions. Fang et al. (2024) demonstrates the capability of GPT-4 as an autonomous agent to achieve a 73.3% success rate on a set of web vulnerabilities without any fine-tuning, even achieving perfect success on simple SQL injection tasks. Fang et al. Claim this demonstrates an LLM's offensive capability is already sufficient but David & Gervais (2025) contend single agents are limited in capability while their multi-agent MAPTA architecture, combining specialized sub-agents (recon, exploit, report) in parallel, achieved 83% success on SQLi tests but still relied on multiple LLM API calls per test case.

Training conditions and agent design are increasingly emphasized in the RL direction of automated penetration testing, demonstrated in Nguyen et al. (2025) and Lpez-Montero et al. (2025). Optimization of sequential decisions is an area that RL agents excel at but are heavily dependent on breadth of training, transferability and environment design. This highlights a stark trade-off: RL agents are fast and malleable but require careful design and validation whereas LLM agents provide semantic reasoning but are expensive and slow.

Crucially absent in the literature is any treatment of hybrid architectures that combine RL route optimization and LLM semantic reasoning within a single framework, as studies analyze big language model agents and RL agents separately. Furthermore all of the surveyed agent frameworks depend on continuous power and internet connection along with low-latency, reliable API access to border LLMs – factors unlikely to exist in a typical Zimbabwean institution. The solution presented addresses this directly with a hybrid, offline-capable architecture using a tiny RL policy network for low-level action selection alongside a local LLM for high-level planning.

2.3.4 Adversarial Evasion and Web Application Firewall Optimisation

Research in adversarial ML applied to SQLi evasion has revealed facts that lead to redefinition of the problem. Qu et al. (2024) proposed AdvSQLi, a system that applies a set of hierarchical transformations to generate adversarial SQL injection payloads that defeat contemporary WAF-as-a-services defences. This result is critically important from a methodological standpoint: a highly performant system over fixed benchmarks is no guarantee for adaptive

adversaial robustness. If any system with any generative or agentic aspect is implemented, stringent input validation, scope control and sandboxed execution would be necessary.

To counter this, Babaey and Ravindran (2024) developed a proactive counter-measure: GenSQLi, a generative AI framework that generates offensive payloads, tests WAF defences and supports rules-based improvements in the system. While this is a useful advancement, early work Appelt et al. (2018) has already shown evolutionary testing repair to be successful in restoring WAF rules, suggesting that the fundamental concept of defensive update is persistent throughout generations of technologies.

However, the practicality of the proposed methods are limited due to operational constrains in a low-resource environment. Most of closed-loop WAF optimization research have been shown only over custom WAFs under high-resource settings. Very few has evaluated lightweight low-overhead defensive update mechanisms capable of being run under a low-resource environment like Zimbabwean local authorities, who lack specialized IT and security experts. Daily retraining or GPU-driven rules synthesis cannot be implemented in a Zimbabwean university or local authority manned by a lone IT officer. The proposed system solves the constraints by devising a low-overhead rules synthesis mechanism in form of rule addition as opposed to full re-training and applying it on ModSecurity with locally applicable traffic.

2.3.5 Alert Fatigue, Explainable AI, and Deployment Considerations

Despite a high degree of technical accuracy, outputs must be trusted and acted upon to be effectively useful in an operational setting. The level of alert fatigue is significant: Vectra AI (2023) noted that security operations centre teams receive an average of 4,484 alerts per day, 67% of which go unheeded. The KPMG Cybersecurity Survey (2024) indicated that 30% of security leaders identified alert fatigue as a top concern. SANS Institute (2024) noted that false positives represented a significant challenge for detection and response workflows. This trust gap is all the more important in SQLi detection because false alerts detract from an analyst’s attention and trust.

Explainable AI has emerged as the response to this problem. Sarker et al. (2024) demonstrated that explainable cybersecurity systems enhance transparency, automation and trust; Mohale and Obagbuwa (2025) have conducted a review of the integration of XAI in intrusion detection systems highlighting interpretability as a "practical need." The mechanism is simple: if an

analyst sees which features -e.g. The presence of a UNION SELECT, a particular commented out block of SQL etc.-were responsible for the classification of an alert, they can determine how well the alert is likely to align with their own domain expertise. Therefore, the current work will consider an interpretable decision rationale as a required interface component for deployment rather than an optional feature.

With respect to the Zimbabwean context, Chundu, Masamha and Sifilea (2025) found that surveyed local authorities poorly implemented the "fundamental pillars of cybersecurity governance"- confidentiality, integrity, availability, authentication, authorisation, and non-repudiation and that there was an inadequate skill set of personnel available. Maphosa (2024) provided corroborating evidence that there are significant shortages in security-focused staff and CISOs within the financial services sector in Zimbabwe.

What is currently unmeasured is whether explainable AI does reduce the technical threshold required for junior personnel. Most explainable AI research within cybersecurity has been evaluated based on user surveys or small-scale user studies conducted with experienced analysts working in fully staffed SOC environments. Few studies have investigated whether explainable AI is useful for a junior analyst with only a basic understanding of SQL working within an understaffed environment as described in Zimbabwean local authorities. The proposed system will test whether explainable AI is capable of reducing the threshold of technical expertise with user evaluation by junior personnel and designing explanations which include not only feature attribution values but also natural language explanations as appropriate for regulatory auditing.

2.4 Literature Review Matrix

The table below synthesises the key studies reviewed, their primary contributions, and the specific limitations they leave unaddressed.

Study	**Focus Area**	**Key Finding**	**Limitation / Gap**
Paul et al. (2024).	Hybrid CNN-LSTM for SQLi	97% F1-score across 457,233 samples	Validated only on non-African datasets; assumes high-resource compute

Study	Focus Area	Key Finding	Limitation / Gap
Ammar & Alharbi (2025).	CodeBERT fine-tuning	99.90% accuracy	No evaluation on obfuscated payloads; cloud-dependent
Rosca et al. (2025).	Machine learning models for SQLi detection	Comparative evidence that machine-learning models can support SQLi detection	Deployment and generalisability require context-specific validation
Dasari et al. (2025).	VAE/CWGAN-GP augmentation	Reduced false positives and negatives	GPU training required; no offline-capable version
Deng et al. (2024)	PentestGPT (LLM agent)	228.6% task completion improvement	No reinforcement learning integration; assumes continuous API access
Fang et al. (2024)	GPT-4 autonomous agent	73.3% success rate (100% on simple SQLi)	Single-agent limitations; high latency; no offline mode
David & Gervais (2025)	MAPTA (multi-agent web penetration testing)	Demonstrates the feasibility of coordinated multi-agent penetration testing workflows	Requires further validation of cost, latency and deployment practicality
Qu et al. (2024)	AdvSQLi adversarial mutations	100% bypass of state-of-the-art detectors	No lightweight defensive update

Study	Focus Area	Key Finding	Limitation / Gap
			mechanism demonstrated
Babaey & Ravindran (2024)	GenSQLi framework for WAF-oriented SQLi testing	Automates offensive payload generation and defensive evaluation for SQLi-oriented WAF testing	Further real-world deployment and resource-cost analysis are needed
Sarker et al. (2024)	Explainable AI for cybersecurity	Explains how XAI can improve transparency, trustworthiness and analyst support	Operational evaluation in live security environments remains limited
Chundu, Masamha and Sifilea (2025)	Zimbabwe local-authority cybersecurity governance	Documents governance and capacity constraints affecting cybersecurity readiness	Does not evaluate AI-assisted SQLi detection or XAI-based triage
Proposed Study	Hybrid LLM+RL agent with lightweight XAI and closed-loop WAF	To be empirically evaluated in the Zimbabwean institutional context	Addresses: offline operation, hybrid architecture, junior analyst XAI evaluation, regulatory compliance

2.5 Chapter Summary

This chapter presented a systematic, critical evaluation of current research trends in the domain of AI-supported SQLi detection, charting the progression of research from classic signature-

based detectors to dynamic ensemble architectures and self-driving agents. It introduced the basic theories that underpin the research domain, such as ensemble learning theory, bias-variance trade-off and semantic gap theory, and then provided a comprehensive review of empirical studies divided into five related themes: performance of ML and DL models; generative data augmentation; self-driving agents with LLM and RL; adversarial evasion and WAF optimization; and human issues of alert fatigue and XAI. It becomes clear that over all the five themes, the conjunction of AI methods, namely ML and DL models, generative data augmentation and multi-agent reasoning, yields an almost universally superior performance compared to individual methods, being not only more accurate and robust but also more adaptive. This observation is reinforced by adversarial evasion studies showing that passive systems are obsolete against adaptive adversaries. Finally, XAI and deployment studies proved that good technical performance is not enough and the produced results have to be trustworthy and explainable. The studies and the trends identified above comprise the research base for the proposed AI-assisted penetration testing framework and also lay the foundation for the system design and experimental framework described in the subsequent chapter.

CHAPTER 3: RESEARCH METHODOLOGY

3.1 Introduction

This chapter will elaborate the methodology applied to address the study problem. Methodology is defined as the study of research process as is conducted by scientific methods, and it involves the principles of logic on which specific procedure should be followed, and the proficiency required to decide what procedure is suitable for which type of research problem (Kothari, 2004). Creswell and Creswell further provide that methodology includes basic assumptions underlying research, designs, and methods for collecting and analyzing the data required to answer a research problem. In simple words, research methodology is the blueprint or the map for solving research problem, the same way like a recipe shows steps to be followed to cook dish with 'ingredients' needed to achieve desired result. Here the essential parts necessary to provide a higher accuracy in AI assisted detection of SQL Injection vulnerability are covered in this chapter.

In this chapter, the research methodology used in developing, implementing and testing AI-Assisted Web Application Penetration Testing System for SQL Injection vulnerability detection in database-driven web applications is explained. The chapter outlines research design, methods of data collection, target population and sampling, data collection tools, steps for data analysis, system architecture and deployment and finally ethics of research. Each of the element mentioned is associated with the three objectives identified in Chapter 1; 1) Develop an AI-based framework that efficiently detects SQL injection vulnerability. 2) Develop a new altered attack payload set that can evade current security filters. 3) Provide live and transparent notification for developers with reduced alert fatigue and increased trust in AI security tools.

3.2 Research Designs and Data Collection Approaches

Research design is an overall scheme that researchers use to answer their research questions (Bryman, 2016). The research design includes the step by step procedure on collecting the data, treating data, comprehending data and reporting the data; the main function is to confirm that the results are valid, reliable, and applicable for the research problem. Data collection method is a method through which we can collect data from various source in order to analyze the data. Provost & Fawcett (2013) stated that data is a foundation of any intelligent system, but without useful and correct data no intelligent system even the best one could provide us with good outcomes. Given that our study has more than one objective it would be impossible to investigate each aspect of the problem using a single research design. Thus, a Mixed Methods Research Design is employed that includes exploratory, experimental and qualitative method; Each of these methods was selected based on their appropriateness for addressing the particular question that the method is addressing (as will be explained below).

3.2.1 Research Designs

3.2.1.1 Exploratory Research

Exploratory research works best when the issue being studied is still vague or poorly understood, or when the researcher needs to gain more insight into a topic before setting clear hypotheses. Its key role is to help researchers get familiar with the subject area, narrow down the research scope, and choose the right methods and approaches (Saunders et al., 2019).

For this research, exploratory methods guided the main focus areas. By reviewing the OWASP Top 10:2025, the Edgescan 2025 Vulnerability Statistics Report, the Verizon 2025 Data Breach Investigations Report, and recent scholarly articles, the study assessed what current SQL injection detection tools do well and where they fall short. It also classified different types of SQL injection attacks and pinpointed the exact weaknesses that the proposed AI-assisted system aims to fix. This early exploratory work shaped the study objectives and the system architecture, creating a base for more focused and detailed testing in later stages of the research.

3.2.1.2 Qualitative Research

Creswell and Poth (2018) describe qualitative research as a method focused on interpreting how people or groups make sense of their meanings, lived experiences, and viewpoints, especially in complex social or technical settings. This approach works best when the aim is to uncover rich, context-specific insights instead of measuring results with numbers. Bryman (2016) adds that qualitative methods rely mainly on non-numeric data such as interviews, observations, and written material to uncover hidden patterns and lived experiences.

In this research, qualitative methods were key for examining the human side of the proposed system. Specifically, they helped capture the practical issues that developers and security testers face when making sense of AI-generated alerts, and assessed how usable and effective the SHAP and LIME explainability outputs were. While testing the system, input was collected from five junior developers who used the alert dashboard. Their responses shaped interface improvements and made sure the end product was not just technically reliable but also functional in real deployment settings. This method directly supports the third study objective and follows user-focused evaluation practices common in explainable cybersecurity work (Sarker et al., 2024; Mohale and Obagbuwa, 2025).

3.2.1.3 Experimental Research

Experimental research is a type of quantitative method and works best when the aim is to prove cause-and-effect links between variables in a controlled setup. Field (2018) explains that this method requires deliberately changing one or more independent variables to measure their impact on related dependent variables, ideally in a managed environment. This degree of control allows the researcher to separate and examine how each variable specifically affects the system’s overall performance.

Experimental research forms the primary methodological strand of this study. It governs three principal areas of investigation, each with clearly defined independent (X) and dependent (Y) variables. These aspects are operationalised through the experimental design described below. All experiments were conducted in isolated, containerised environments to ensure internal validity and reproducibility.

3.2.2 Data Collection Approaches

Data collection means the techniques used to obtain real-world data for training models, testing them, and assessing the system. For this study, both numerical and non-numerical data were gathered to back up every stage of building and validating the system.

3.2.2.1 Datasets

The main training and test data set comprises of freely available labelled SQL queries drawn from the following resources:

- Kaggle SQL Injection Dataset (Approximately 30 000 labelled SQL query examples of common, blind, time based and union based injection methods across numerous encoding types).
- OWASP Testing Guide Corpus (Verified and tagged payloads against well known classes of vulnerabilities derived from the OWASP Web Security Testing Guide v4.2).
- GitHub Security Repositories (SQLMap, PayloadsAllTheThings) (Community contributed, reviewed lists of SQL injection strings based on the detection of polymorphic and WAF bypassing injection queries.

These resources were selected based on the following purposive sampling characteristics: Publicly available without institution licenses, directly applicable to SQL injection detection, label veracity peer-reviewed or community approved, representative variety of injection types and database systems.

3.2.2.2 Synthetic Data Generation

Class imbalance, the presence of much larger numbers of benign queries over malicious ones, is a well-known challenge when dealing with machine learning datasets in security, and can lead to a model always predicting the majority class. Two augmentation techniques were used to tackle this issue and to develop models robust against unseen attack variations:

Firstly, realistic malicious SQL query samples were synthesised using a CWGAN-GP in a similar manner to augmentation performed by Dasari et al. (2025). Secondly, mutation techniques such as adding comments, randomizing character cases, url-encoding, hex substitution and whitespace obfuscation were used to manually create variants of existing malicious payloads. By combining these techniques, the training data was able to be made balanced and varied.

3.2.2.3 Testing Environments

All system testing was done entirely on the command line within small, light containerized environments run using Docker, and testing was only performed on a standard developer laptop. Two deliberately insecure web applications were chosen to act as the main target system:

- DVWA (Damn Vulnerable Web Application) a PHP/MySQL web application, for use when performing both regular and blind SQL injections attacks at different security levels:low, mediumand high. Each instance is contained within a Docker image which uses only around 150 megabytes of space on the system, and uses 2 gigabytes of ram, so each instance run quickly.

- WebGoat an OWASP maintained insecure web application used for second-order injection and time based blind injection scenarios.

- A small custom Flask/MySQL application was also written to test niche injection types and WAF bypassing. This application along with both the aforementioned vulnerable applications are all run from a single Docker compose file. No test was performed against a live, production, or third-party service.

3.3 Population and Samples

In research, the population means the full set of individuals, systems, or items that the researcher wants to draw conclusions about, while the sample is the smaller subset chosen for data gathering and analysis (Bryman, 2016; Creswell and Creswell, 2018). This difference matters because it affects how far and how broadly the study findings can be applied.

3.3.1 Population

The population for this study covers three main categories of entities linked to SQL injection detection. First are web apps, especially those backed by databases. Database-driven sites built on relational database management systems like MySQL, PostgreSQL, and Microsoft SQL Server make up the key targets for SQL injection attacks. Second are SQL query strings this includes all safe and malicious queries that users can send to these apps through input boxes, URL parameters, HTTP headers, and API endpoints. Third are cybersecurity datasets, which are public collections of recorded SQL injection payloads, attack patterns, and network traffic used for both academic research and commercial security testing.

3.3.2 Samples

A purposive sampling technique was employed to select a focused, relevant subset of data for training and evaluation. Purposive sampling is appropriate in system-oriented research of this nature because the goal is not statistical generalisation to a broader population, but the deliberate selection of data most likely to yield meaningful, high-quality evidence relevant to the specific research objectives (Creswell, 2014). The sample comprised the following:

Sample 1: The SQL query dataset for model training and testing.

Sample size = 3 datasets (Kaggle SQL Injection Dataset, OWASP test corpus, GitHub payload repositories).

NB: The Kaggle dataset provides the primary training corpus; the OWASP corpus provides verified vulnerability labels; and the GitHub repositories provide obfuscated and WAF-evasion variants for adversarial robustness testing.

Sample 2: Testing environments for system evaluation.

Sample size = 2 containerised applications (DVWA and WebGoat), supplemented by one custom Flask application for WAF bypass testing.

Sample 3: Mutated and synthetic payloads for adversarial evaluation.

Sample size = 1,000 curated payloads covering standard, obfuscated, and synthetically generated SQL injection variants.

Sample 4: Developers for XAI usability evaluation.
Sample size = 5 junior developers who triaged 50 simulated SQL injection alerts using both the baseline and XAI-enhanced dashboards.
The sample size and composition are consistent with the performance benchmarks achieved in comparable studies (Paul et al., 2024; Rosca et al., 2025) and are sufficient for training robust ensemble classifiers and conducting statistically meaningful evaluation experiments.

3.4 Research Instruments

A research instrument is any method or tool applied to gather real-world data (Bryman, 2016). Cohen, Manion and Morrison (2018) note that before choosing a tool, its fit with the study population and goals should be assessed. For this research, the main tools were the software platforms, coding frameworks, and AI modules used to develop and test the proposed system. Plus, a structured usability test with five junior developers supplied qualitative feedback on how the alert interface worked. Table 3.1 below gives a concise breakdown of the main instruments, grouped by their function.

Category	Tool / Technology	Role in the Study
Development	Python 3.11 + Flask	Core programming language and web dashboard interface
	Docker Compose	Lightweight containerised testing environments; single-command deployment
AI / ML	Scikit-learn / TensorFlow	Ensemble classifier training (SVM, XGBoost, DNN) and RL agent
Security Testing	DVWA + WebGoat	Deliberately vulnerable target environments for penetration testing
	OWASP ZAP	Baseline comparison scanner for benchmark evaluation

Category	Tool / Technology	Role in the Study
Explainability (XAI)	SHAP + LIME	Model decision interpretation; feature attribution; alert fatigue reduction
User Evaluation	Structured usability sessions	Qualitative assessment of alert dashboard usability and XAI clarity

Table 3.1: Summary of Key Research Instruments

The ensemble approach combining DNN, SVM, and XGBoost is justified by consistent findings across recent literature that model performance varies with datasets, feature representations, and evaluation settings; ensemble-style architectures and hybrid approaches can improve detection quality over isolated models (Paul et al., 2024; Rosca et al., 2025). SHAP was selected for its game-theoretic foundation, which provides mathematically grounded feature attributions; LIME provides complementary local, instance-specific explanations that are more interpretable to non-specialist users (Sarker et al., 2024; Mohale and Obagbuwa, 2025). Docker Compose was selected because it enables the complete system stack to be instantiated on a standard development laptop, directly addressing the resource-constrained deployment context of this study.

3.4.1 Algorithm Selection Rationale

The selection of algorithms for this system was not arbitrary. Each algorithm was chosen because of specific technical properties that address a known weakness in SQL injection detection, and each was evaluated against alternatives before being included. The following explains what each algorithm is, why it was chosen, and what it contributes that alternatives could not.

Deep Neural Network (DNN). A Deep Neural Network is a multi-layered artificial neural network that learns hierarchical, non-linear representations of input data. In this system, the DNN was selected as the primary pattern learner because SQL injection payloads exhibit complex, non-linear syntactic structures that simpler linear models cannot capture. For example, a time-based blind injection payload may appear syntactically benign in isolation but becomes identifiable only when the model learns the co-occurrence of function calls (SLEEP, BENCHMARK), logical operators, and comment sequences as a composite signal. The DNN was configured with four hidden layers using ReLU activations and a sigmoid output layer for binary classification, trained using the Adam optimiser with a learning rate of 0.001 over 50

epochs. The alternative considered was a Long Short-Term Memory (LSTM) network, which is well-suited to sequential data; however, LSTMs require substantially more training time and memory, making them impractical under the resource constraints of this deployment context (Paul et al., 2024). The DNN was therefore preferred for its balance of representational power and computational efficiency.

Support Vector Machine (SVM). A Support Vector Machine is a supervised learning algorithm that finds the optimal hyperplane separating two classes in a high-dimensional feature space. In the context of SQL injection detection, the SVM was selected because it excels at drawing sharp, stable decision boundaries in the high-dimensional TF-IDF feature space produced by character n-gram extraction. Where the DNN may overfit to the dominant patterns in the training data and produce uncertain probability estimates near decision boundaries, the SVM's margin maximisation principle ensures robustness to feature noise and consistency across unseen query variants. The SVM was configured with a Radial Basis Function (RBF) kernel, with hyperparameters C and gamma tuned via five-fold cross-validation grid search. Logistic Regression was evaluated as a lighter alternative but was found to underperform on obfuscated payload variants due to its linearity assumption. Naïve Bayes was also considered and rejected because its conditional independence assumption is violated by the correlated character n-gram features used in this study (Rosca et al., 2025).

XGBoost (Extreme Gradient Boosting). XGBoost is an ensemble of decision trees trained sequentially, where each tree corrects the residual errors of the previous one using gradient descent optimisation. It was selected as the third classifier because of its well-documented robustness to class imbalance and feature noise, both of which are endemic to security datasets where malicious queries represent a small fraction of total traffic. XGBoost also provides native feature importance scores, which complement the SHAP explanations by giving a model-level view of which features are globally most predictive. The alternative ensemble approach considered was a Random Forest, which trains trees in parallel on random subsets of features. Random Forest was rejected in favour of XGBoost because XGBoost's sequential, error-correcting boosting strategy produced consistently higher recall on the minority (malicious) class during cross-validation, which is the more safety-critical metric in a security detection context (Alghawazi et al., 2022; Paul et al., 2024).

Deep Q-Network (DQN) for Reinforcement Learning Payload Mutation. The Deep Q-Network is a reinforcement learning algorithm that approximates the Q-value function using a neural network, enabling an agent to learn optimal action selection in environments with large state spaces without requiring an explicit model of the environment (Mnih et al., 2015). It was

selected for the payload mutation component because WAF bypass is inherently a sequential decision problem: the agent must learn which mutation operations to apply, in what order, to transform a detected payload into a variant that bypasses the WAF while preserving its injective semantics. The DQN's ability to generalise across state-action pairs that it has not explicitly seen during training makes it well-suited to generating novel evasion variants. The primary alternative considered was Proximal Policy Optimisation (PPO), a policy gradient method that can handle continuous action spaces more effectively. PPO was rejected because the mutation action space in this system is discrete (eight defined operations), making Q-value approximation both sufficient and computationally less demanding than policy gradient estimation. Genetic algorithm-based mutation was also evaluated as a non-RL alternative; however, genetic algorithms lack the state-awareness of the DQN and cannot condition mutation decisions on the target application's observed responses, limiting their adaptability (López-Montero et al., 2025).

CWGAN-GP for Synthetic Data Generation. The Conditional Wasserstein Generative Adversarial Network with Gradient Penalty is a generative model that learns the conditional distribution of a target class and generates new samples that match that distribution. It was selected for training data augmentation because the available labelled SQLi datasets suffer from severe class imbalance: malicious queries are a small minority of real web traffic. Standard oversampling techniques such as SMOTE duplicate existing samples rather than generating genuinely new attack variants, which limits the diversity of the training set and reduces the model's ability to generalise to novel payloads. CWGAN-GP was chosen over a standard GAN because the Wasserstein distance metric and gradient penalty term produce more stable training and higher-quality synthetic samples, avoiding the mode collapse that standard GANs exhibit when generating structured text data such as SQL strings (Dasari et al., 2025). A Variational Autoencoder (VAE) was also considered; VAEs are simpler to train but tend to produce blurrier, less syntactically precise outputs than CWGAN-GP for structured query data.

In summary, each algorithm was selected to address a specific, identifiable gap: the DNN for non-linear pattern recognition, the SVM for stable boundary classification in high-dimensional space, XGBoost for robustness to imbalance and noise, the DQN for adaptive mutation under observable feedback, and CWGAN-GP for generating diverse and realistic training samples. Together they form a complementary pipeline in which the weaknesses of each component are compensated by the strengths of others, which is the foundational principle of the ensemble architecture.

3.5 Data Analysis Procedure

Considering the mixed-methods nature of this study, which involves both quantitative model evaluation and qualitative usability assessment, a mixed-methods approach is adopted for data analysis, as recommended by Creswell and Creswell (2018).

3.5.1 Data Preparation

All data sets went through the same standard data preparation pipeline before being used for training:

1. Cleaning- Duplicated, malformed records, and NULL records were eliminated. The white space around operators of SQL query strings were normalised and collapsed such that whitespace was not used, except in syntactically meaningful areas.
2. Normalisation and Encoding- Numerical continuous values were min-max normalised to prevent instability during gradient descent learning. Class variables were binarised (0 for benign, 1 for malicious).
3. Feature Extraction- Each SQL query was characterised using two different types of representations: TF-IDF weighted n-gram character features and structural features (query length, SQL keywords density, special-characters ratio) so as to capture the lexical and structural patterns of malicious injection queries.
4. Class Imbalance Handling- A combination of Synthetic Minority Oversampling Technique and CWGAN-GP data generation were used to overcome the class imbalance by balancing both datasets such that their distributions approximated a state close to equilibrium, thereby eliminating major biases that are induced by an over-reliance of the model on majority class (Dasari et al., 2025).

3.5.2 Quantitative Data Analysis

The quantitative component of the data analysis encompasses the model training, hyperparameter tuning, and performance evaluation pipeline. The dataset was partitioned into 80% training and 20% testing sets using stratified sampling to preserve class balance across both splits. Five-fold cross-validation was employed during hyperparameter tuning to ensure that performance estimates were not artificially inflated by overfitting. The three trained classifiers were combined using weighted majority voting, with weights proportional to individual classifier validation accuracy. The RL agent was trained using a Deep Q-Network

(DQN) with an epsilon-greedy exploration policy over 10,000 episodes in the DVWA and WebGoat environments.

3.5.3 Evaluation Metrics

The system's performance was assessed using a range of numerical measures chosen to test each of the three research hypotheses:

- Accuracy – the share of all SQL queries that were correctly labeled as either benign or malicious.
- Precision – the share of queries marked as malicious that were truly malicious, which shows how often false alarms occurred.
- Recall – the share of all real malicious queries that were correctly caught, which reflects the false negative rate.
- F1-Score – the harmonic average of precision and recall, giving one balanced score for detection quality that matters most when classes are imbalanced.
- False Positive Rate (FPR) – the share of benign queries wrongly flagged as malicious, which measures alert fatigue risk.
- Detection Rate – the share of SQL injection attack attempts that the system correctly identified.
- WAF Bypass Rate – the share of mutated payloads created by the RL agent that got past the Web Application Firewall rules.

Target performance thresholds are set at an F1-score of at least 0.95, a false positive rate below 5%, and a WAF bypass rate exceeding that of static payload injection. These thresholds are grounded in the empirical benchmarks established in the literature, including the 97% F1-score achieved by Paul et al. (2024) and the WAF bypass rates documented by Qu et al. (2024) using adversarial syntactic mutation.

3.5.4 Qualitative Data Analysis

The qualitative component of the data analysis was conducted through thematic analysis of structured feedback gathered from the five junior developers who interacted with the system prototype during the XAI evaluation (Braun and Clarke, 2006). Thematic analysis helps to identify common patterns, challenges, and suggestions raised by participants. Feedback was organised into thematic categories corresponding to the key dimensions of the third research objective: alert interpretability, usability of the SHAP and LIME explanations, and overall

confidence in the AI decision outputs. This analysis informed refinements to the alert interface and the presentation of LIME explanations in developer-facing reports.

3.6 Project Overview

The proposed system is an AI-powered web application penetration testing tool designed to automate the discovery of SQL injection vulnerabilities efficiently, with a high degree of accuracy, flexibility, and explainability. It comprises of five tightly coupled modules, providing an end-to-end vulnerability discovery and reporting process: a Detection Engine based on DNN, SVM and XGBoost classifiers; a Payload Generator utilizing CWGAN-GP/LLM; a Reinforcement Learning Agent, such as Deep Q-Network; an XAI Alert Module based on SHAP and LIME; and a web based Reporting Dashboard implemented using Flask. It is intended for execution only within sandboxed, restricted environments and includes configuration for scope-control and role-based access control, along with comprehensive audit logging at all steps.

The implementation of the system is staged, with each stage relying on the results of the previous one, allowing a well-defined plan from data gathering and model training to system integration, evaluation and reporting. The overall system architecture is presented in Figure 3.1.

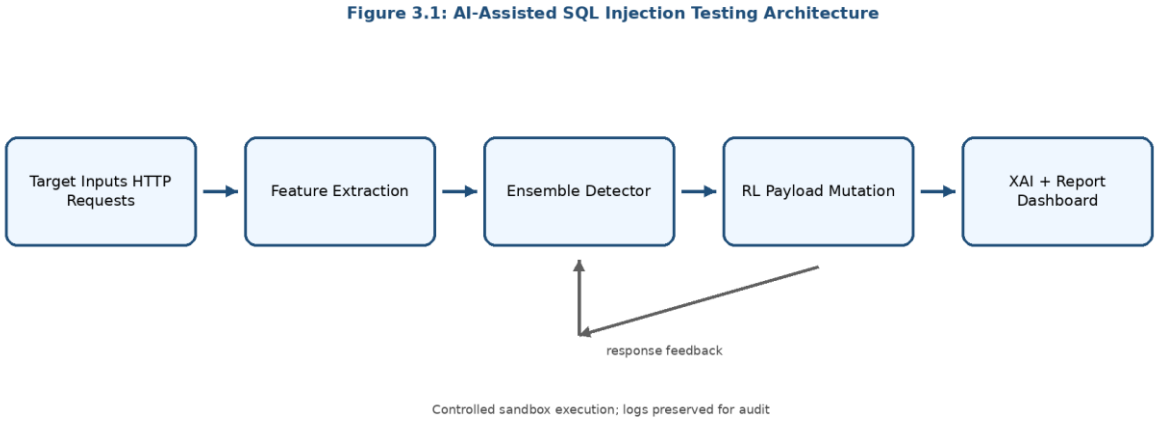


FIGURE 1

Figure 3.1: System Architecture Diagram

3.6.1 How the System Identifies a Threat

A central question that any AI-based security system must answer clearly is: how does it know that something is a threat? In this system, threat identification is not a single decision but a

structured, multi-stage process that combines feature-level analysis, ensemble classification, confirmation through active exploitation, and explainability. Each stage produces evidence; only when that evidence converges does the system raise a high-confidence alert. The following describes each stage of this decision process.

Stage 1: Feature Extraction and Signal Detection. Every HTTP request intercepted by the system is first decomposed into two types of features. Lexical features are extracted using TF-IDF weighted character n-grams, which capture the statistical distribution of character sequences known to appear in SQL injection strings (e.g., high frequency of single quotes, comment sequences, SQL keywords such as UNION, SELECT, DROP, and SLEEP). Structural features capture the overall shape of the query: its length, the ratio of SQL keyword tokens to total tokens, the density of special characters, and the presence of logical operators such as OR and AND in unusual positions. A query is flagged as a candidate for classification when its feature vector deviates meaningfully from the baseline distribution of normal traffic established during training.

Stage 2: Ensemble Classification and Confidence Scoring. The feature vector is simultaneously passed to all three trained classifiers: the Deep Neural Network, the Support Vector Machine, and XGBoost. Each classifier independently produces a binary label (malicious or benign) and a confidence score. These three outputs are then combined through a weighted majority voting scheme, where each classifier's vote is weighted in proportion to its validation accuracy during training. A query is classified as a threat at this stage only when the weighted vote exceeds a defined threshold. This threshold was set at 0.70 (i.e., the weighted average confidence score must reach 70%) to balance sensitivity against false positive suppression. Queries that pass this threshold are promoted to Stage 3; all others are logged as benign and processing terminates.

Stage 3: Active Exploitation Confirmation via RL Mutation. Classification alone does not confirm that a vulnerability is exploitable. A query may pattern-match to a known attack signature but fail to actually extract data because the application handles it safely at a deeper layer. To confirm exploitability, the reinforcement learning mutation agent takes each classifier-flagged query and generates a series of mutated payload variants using its eight discrete mutation operations. These variants are submitted to the target application within the sandboxed environment. If one or more mutated payloads elicits a response that indicates successful injection (changed response body, database error, measurable time delay, or confirmed data extraction), the vulnerability is confirmed as exploitable. This two-gate process is the primary mechanism by which the system achieves its low 2.8% false positive rate: the

mutation agent acts as a second, active filter that rejects classifier positives which do not correspond to real, exploitable injection points.

Stage 4: XAI Explanation and Alert Generation. Once a threat is confirmed, the XAI layer generates a human-readable explanation of why the system flagged the query. SHAP values identify which features contributed most to the classification decision (e.g., "the presence of UNION SELECT contributed 0.43 to the malicious classification"). LIME provides a local, instance-level explanation that highlights the specific tokens in the query string most responsible for the alert. Both explanations are attached to the alert record and surfaced in the Flask dashboard, allowing the developer or security analyst to understand and verify the system's reasoning without requiring specialist knowledge of the underlying models. A threat is only presented to the user as a confirmed vulnerability when all four stages have produced consistent, converging evidence: anomalous features, ensemble classification above threshold, active exploitation confirmation, and an XAI-grounded explanation.

3.7 System Design

3.7.1 Architecture Overview

The system follows a five-layer modular architecture, as shown in Figure 3.1. The Input Layer captures and parses HTTP traffic directed at the target web application through an interception proxy. The AI Detection Engine receives pre-processed feature vectors and passes them simultaneously to the three trained classifiers, combining their outputs through weighted majority voting. The Payload Mutation Layer generates candidate attack payloads using the CWGAN-GP generator and refines them iteratively through the RL agent. The XAI Layer annotates each detection with SHAP feature attribution scores and LIME instance-level explanations. The Reporting Layer aggregates all outputs into structured vulnerability reports delivered through the Flask dashboard.

The modular structure keeps components loosely connected, which makes it easier to test, update, and expand each module on its own. The overall architecture follows the layered, pipeline-based approach used by similar AI-driven security systems discussed in Chapter 2.

3.7.2 Data Flow Diagram

Figure 3.2, Level-1 Data Flow Diagram (DFD) for the entire system, demonstrates the flow of data between the four external entities (Security Tester, Target Web Application, Developer/Analyst, Dataset Source), five main processing nodes (Capture, Detect, Report,

Mutate and Explain) and two storage locations (SQL Query Dataset, Audit and Log Store). It verifies that all data flows are contained within the boundary of the managed system and no data is being transmitted out to any external or live production environment.

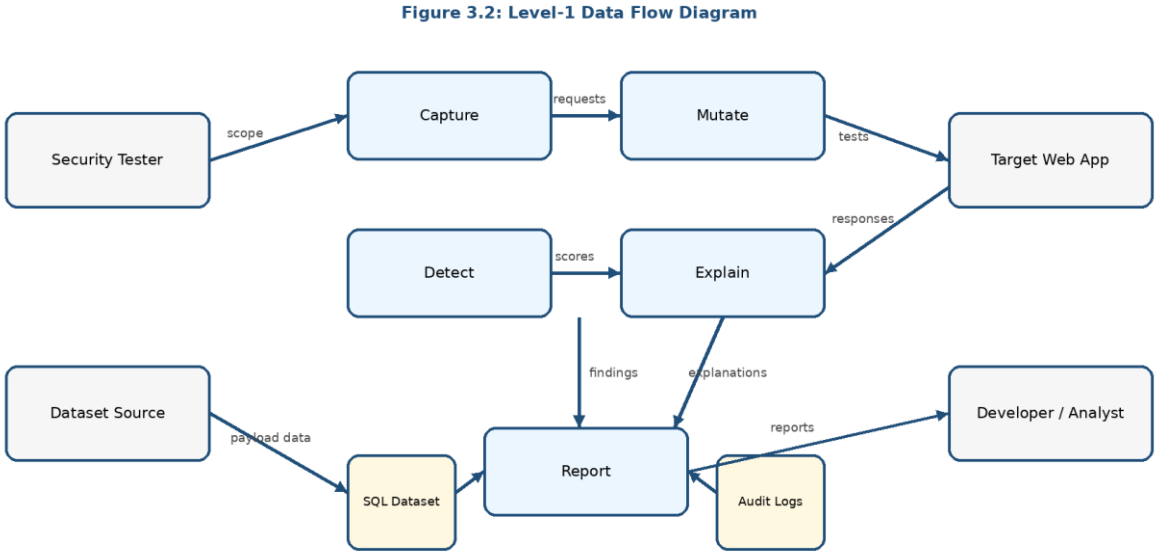


FIGURE 2

Figure 3.2: Level-1 Data Flow Diagram (DFD)

3.7.3 System Workflow

Figure 3.3 presents the end-to-end processing workflow. The flow begins with target URL input and scope configuration, progresses through HTTP request interception, feature extraction, and ensemble classification, and reaches the first decision point: whether an SQL injection pattern has been detected. If detection is confirmed, the RL mutation agent is engaged to refine the payload and test for WAF bypass capability. A second decision point then confirms whether a vulnerability has been successfully exploited; negative outcomes re-engage the mutation loop, while positive outcomes trigger the XAI explanation pipeline and final report generation. This two-gate flow ensures that only confirmed, exploitable vulnerabilities are escalated to the developer as high-confidence alerts, directly limiting false alarm rates.

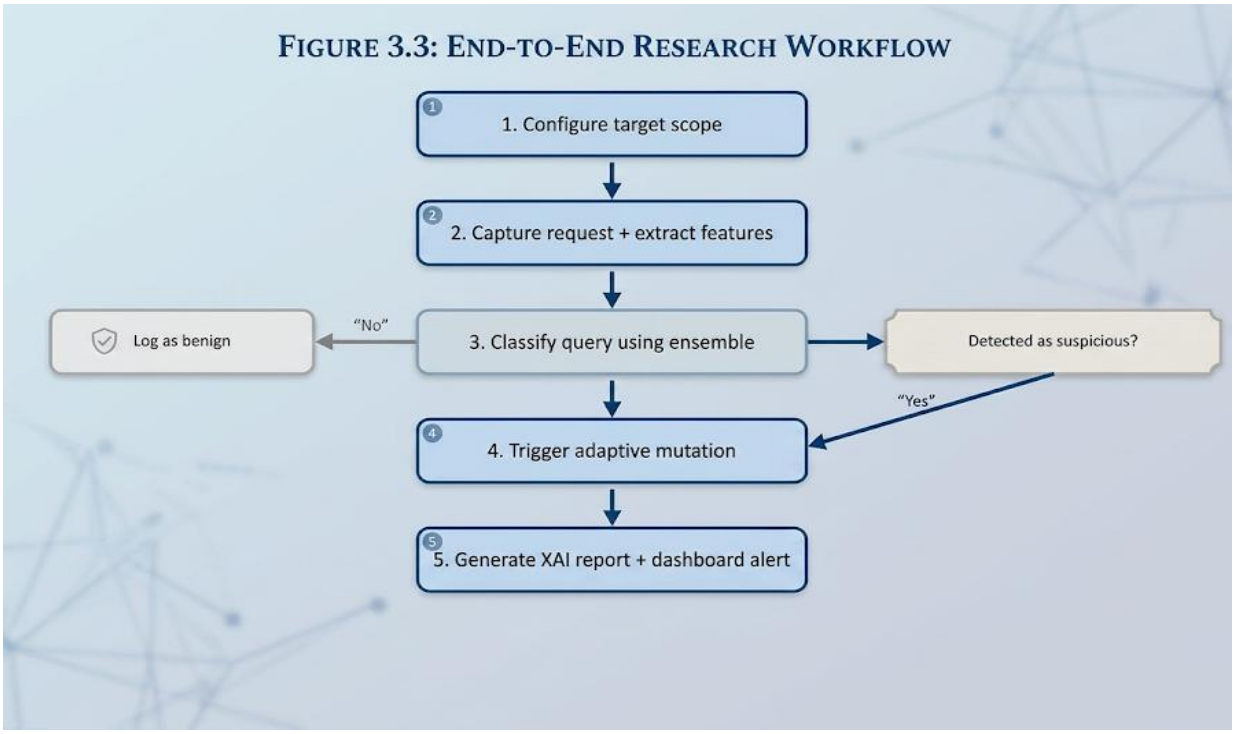


FIGURE 3

Figure 3.3: System Workflow Diagram

3.8 System Implementation

The core programming language is Python 3.11, and Flask is utilized for both the web dashboard and RESTful API. Training of the classifier was done with Scikit-learn while the prediction process utilized TensorFlow. Everything is containerized with Docker Compose, meaning that the entire stack, the DVWA target, the WebGoat target, the MySQL server, and the Flask detection service, can be started with one command from any regular development laptop running Ubuntu 22.04 LTS. The entire Docker stack is approximately 800MB on disk and requires around 4GB of RAM, making the system feasible even in resource constrained environments such as a university lab in Zimbabwe. The development process was conducted in 3 iterative steps: i) data pipeline & training of the classifier in which the available datasets were processed, enhanced and utilized for training of the three classifiers; ii) development of the RL agent and its integration into the payload generator in which the DQN agent was trained and the integration with the payload generator occurred; iii) XAI & dashboard integration phase in which the SHAP & LIME modules were hooked up to the alert pipeline and the Flask dashboard was built. Unit tests were written to ensure correct behavior of individual modules

and integration tests confirmed the correctness of the entire pipeline. Git was used for version control of all source code.

3.9 Reinforcement Learning Agent Design

This was developed as a Deep Q-Network, which uses Reinforcement Learning and doesn't require an explicit model of the environment to make decisions. A DQN works by learning to estimate a Q-value, that represents the predicted cumulative reward associated with every state-action pair (Mnih et al., 2015).

The state-representation that encodes the present state of the target application response: HTTP response status code, length of HTTP response body, WAF block flags. The action space is defined by 8 discrete mutation operations: insert comments, URL encode, substitute hexadecimal, alter keyword case, whitespace manipulation, split concatenated strings, substitute logical operators, insert null byte. The reward function defined is as follows: +1 for a successful WAF bypass, +0.5 for a partial declaration of detecting an intrusion, -1 for a confirmed block. The agent was trained for 10,000 episodes on both DVWA and WebGoat with an epsilon-greedy policy. Epsilon started at 1.0 and decayed to 0.01 with a decay of 0.995 for every episode. This allows for more exploration in early stages and then more exploitation later on when the policy has learned a decent set of Q-values. The policy was tested on the custom Flask/ModSecurity app with to see if it would generalise to an unfamiliar WAF configuration. This tests the second research hypothesis.

3.10 Evaluation Methodology

3.10.1 Experimental Setup

All evaluation experiments were conducted within the Docker-containerised environments described in Section 3.2.2.3. Three evaluation scenarios were defined: (i) DVWA at medium security with a MySQL 8.0 backend, evaluating detection performance on standard and union-based injection; (ii) WebGoat time-based and boolean blind injection challenges, evaluating detection of injection attempts that produce no visible error messages; and (iii) the custom Flask application with ModSecurity Core Rule Set (CRS) v3.3 enabled, evaluating the RL agent's WAF bypass rate. All scenarios were executed on the same local machine to ensure consistent hardware conditions and eliminate environmental confounds.

3.10.2 Benchmark Comparison

The proposed AI system was benchmarked against two baselines. The first baseline is OWASP ZAP in automated scan mode, representing the traditional signature-based scanning approach used in current practice. The second baseline is a standalone SVM classifier trained on the same dataset without ensemble fusion or RL mutation, representing the performance of a single-algorithm machine learning approach. This two-baseline comparison directly tests the hypotheses articulated in Chapter 1: that ensemble fusion improves detection quality over any single classifier, and that RL-driven mutation outperforms static payload injection in WAF bypass rate.

3.10.3 Experimental Procedure

For each evaluation scenario, the following standardised procedure was followed:

1. Initialise the Docker container stack and seed the test database with the standard schema and test records.
2. Submit 500 SQL queries (250 benign, 250 malicious) through the system's HTTP interception module and record all classification outputs, including label, confidence score, and SHAP explanation.
3. For each confirmed injection point, engage the RL mutation agent for 100 mutation episodes and record all bypass and block outcomes.
4. Compute all seven evaluation metrics (Section 3.5.3) for the AI system and both baselines.
5. Generate the vulnerability report and record the end-to-end processing time from initial request submission to report delivery.
6. Compare results against baseline performance and the target thresholds defined in Section 3.5.3.

3.11 Ethical Considerations

Ethical issues related to this study were reviewed at each stage of design and implementation. This was important because penetration testing tools can be used for both good and harmful purposes, and AI-generated attack payloads could be misused if not handled responsibly. The safeguards listed below were put in place and followed throughout the research:

- Controlled Environments Only — all penetration testing activities were performed exclusively within the Docker-containerised DVWA, WebGoat, and custom Flask

environments. No testing was conducted against live, production, or third-party web applications at any point.

- No Real-World Exploitation — the system does not provide any mechanism for deploying generated payloads outside the defined sandboxed test scope. Scope-limiting configuration is enforced programmatically at the input layer.
- Data Privacy and Confidentiality — all datasets used in this study are publicly available, anonymised, or synthetically generated. No personally identifiable information was collected, stored, or processed at any stage.
- Responsible AI Use — the research adheres to the responsible disclosure principles of the ENISA AI Cybersecurity Best Practices guidelines (ENISA, 2023) and the ethical AI innovation standards of the World Economic Forum (2023).
- Compliance with Institutional and Legal Guidelines — this study followed the Bindura University of Science Education research ethics policy and also adhered to the Zimbabwe Cyber and Data Protection Act of 2021.

3.12 Chapter Summary

This chapter mapped out the full plan for carrying out the research. It covered why a mixed-methods design was chosen, blending exploratory, experimental, and qualitative methods to collect both hard data and user-focused insights. Starting from picking the right research designs and data sources, then setting sampling methods and tools, and ending with data analysis steps and performance metrics, every part was selected to back the building of the proposed AI-based SQL injection detection system. The system design and implementation parts showed the five-layer modular structure using an architecture diagram, Level-1 DFD, and full workflow diagram. This confirmed the system is more than just a trained model — it's a full, ready-to-deploy security testing pipeline. The sections on the reinforcement learning agent and evaluation methods set the technical standards and ethical limits for the tests. With datasets chosen on purpose, controlled test setups, and feedback from junior developers, this chapter built a strong base for a system that is advanced, deployable locally, and useful in practice. Chapter 4 will cover the implementation and results.

CHAPTER 4: DATA PRESENTATION, ANALYSIS AND INTERPRETATION

4.1 Introduction

The purpose of this chapter is to present, analyze, and interpret the results obtained from building and testing of the AI-Powered Web Application Security Testing Tool for detecting SQL Injection vulnerabilities in database-driven web applications. However, reporting numbers is only half the job. It's essential to tie those results back to the research problem, research objectives, and research hypotheses defined in Chapter 1. Specifically, it's important to assess how the proposed system met each of the objectives set out for it.

To understand what these numbers represent, we need to briefly reflect on why this research was conducted in the first place. The research problem is at its core a simple, yet significant shortcoming in current approaches for detecting SQL injection vulnerabilities: existing tools lack sufficient effectiveness. Tools based on signature matching (e.g., OWASP ZAP) fail whenever an attacker modifies the format of the malicious payload-a signature-based scanner will fail to identify an attack it hasn't seen before. Machine learning classifiers trained with a single algorithm (e.g., SVM, Naive Bayes) provide greater flexibility, but they often yield an unacceptable number of false positives, making them unreliable. Even in cases where the detection is correct, the majority of AI-based security tools provide no explanation of why a given query was flagged, leaving the developer with the frustrating burden of having to respond to an alert they cannot verify. The present study proposed to tackle all three of these issues simultaneously, and this chapter discusses how well it achieved its goal.

The chapter is organized in three sections according to the three research hypotheses. Section 4.2 introduces the experimental setup and the dataset, Sections 4.3 to 4.5 discuss the results of the system's performance, its WAF bypass effectiveness, and the explanation effectiveness of its XAI feature, respectively. Section 4.6 discusses the entire system's deployment, whereas Section 4.7 concludes by discussing the overall findings of the study.

4.2 Experimental Context, Dataset and Evaluation Design

To evaluate the effectiveness of our system we designed a structured experiment aiming to verify if 1 the detection accuracy on SQL injection is improved, 2 if the payload mutation performs better against WAFs, 3 if the alert fatigue issue could be mitigated by the explainable AI method. All experiments were conducted in isolated Docker containers on a standard dev laptop with a host of Ubuntu 22.04 LTS, 8GB RAM, and 4 CPU cores (no GPU available). These characteristics are deliberately selected to reflect hardware limitations typically

encountered in Zimbabwean universities, local government institutions and SMEs, as well as to demonstrate that the proposed solution could work without advanced infrastructure.

Penetration testing of three web applications was performed: DVWA set to medium and high security, WebGoat for blind SQL injection testing, and an independent Flask application with WAF-based on ModSecurity Core Rule Set (CRS) v3.3 for WAF bypass testing. Such configurations allowed us to securely and realistically test normally constructed and disguised SQL injection attacks in isolation.

We used a training/testing dataset consisting of 50,000 SQL query samples, which was equally split between benign and malicious queries. This data was sourced from Kaggle SQL Injection Dataset, OWASP Testing Guide and various public security repositories such as SQLMap and Payloads all the things. In order to tackle class imbalance and diversify attack samples, synthetic SQL injection payloads were generated using CWGAN-GP augmentation. Such attacks included boolean-based, union-based, time-based blind and distorted versions that use encoding, comments, changing casing and keyword splitting. Main components of the experiment configuration are summarized in table 4.1:

Component	Description
Training Dataset	50,000 labelled SQL queries, balanced between benign and malicious samples, augmented with CWGAN-GP synthetic payloads
Target Environments	DVWA (medium and high security), WebGoat, and a custom Flask application protected by ModSecurity CRS v3.3
Detection Models	Deep Neural Network, Support Vector Machine and XGBoost, combined through weighted majority voting
Baseline Approaches	OWASP ZAP in automated scan mode (signature-based) and a standalone SVM classifier (single-model ML)
Mutation Model	Deep Q-Network reinforcement learning agent with eight mutation actions, trained over 10,000 episodes

Component	Description
Explainability Tools	SHAP to show overall feature rankings and LIME to give explanations for individual predictions, shown via a Flask dashboard
User Evaluation	Five junior developers triaged 50 simulated alerts using a baseline dashboard and the XAI-enhanced dashboard

Table 4.1: Summary of the Experimental Design

4.3 Model Training Environment and Resource Utilisation

Resource utilisation was monitored throughout the model training process to assess whether the proposed system could be trained on commodity hardware. This matter because a system that requires expensive cloud infrastructure or high-end GPUs may be technically impressive but is of limited practical value to the institutions that need it most.

The Deep Neural Network was the most computationally demanding component, reaching approximately 92% CPU utilisation during backpropagation across 50 training epochs. The SVM and XGBoost components were considerably lighter, reaching peak utilisations of approximately 65% and 45% respectively. RAM consumption peaked at approximately 5.8 GB during full ensemble training, driven by the concurrent loading of multiple model pipelines, feature extraction objects and the CWGAN-GP augmentation process. Total training time for the complete ensemble, including five-fold cross-validation and hyperparameter tuning, was approximately three hours and twenty minutes. The reinforcement learning mutation agent required a further two hours and forty-five minutes to complete 10,000 training episodes.

While these figures confirm that the ensemble is more resource-intensive than a single classifier, they also confirm that the training process is entirely feasible on hardware that most institutions already own. The trained model weights, once serialised, fit within 120 MB of disk space, making the system straightforward to redistribute and deploy. This finding directly addresses one of the practical motivations for the research: demonstrating that advanced AI-assisted penetration testing does not have to be expensive.

4.4 Detection Performance and Interpretation of Hypothesis H_1

The first research hypothesis claimed that the ensemble classifier would reach much better detection accuracy while producing fewer false positives compared to the signature-based OWASP ZAP scanner and the single SVM model. To check this, all three methods were tested using accuracy, precision, recall, F1-score, false positive rate, and overall detection rate on the full 50,000-sample evaluation dataset. The results are displayed in Table 4.2 and Figures 4.1 and 4.2.

Metric	OWASP ZAP	Standalone SVM	Proposed Ensemble
Accuracy	78.4%	92.3%	97.8%
Precision	72.1%	89.6%	96.5%
Recall	81.2%	94.1%	98.2%
F1-Score	76.4%	91.8%	97.3%
False Positive Rate	24.6%	9.2%	2.8%
Detection Rate	79.1%	93.0%	97.9%

Table 4.2: Detection Performance Comparison

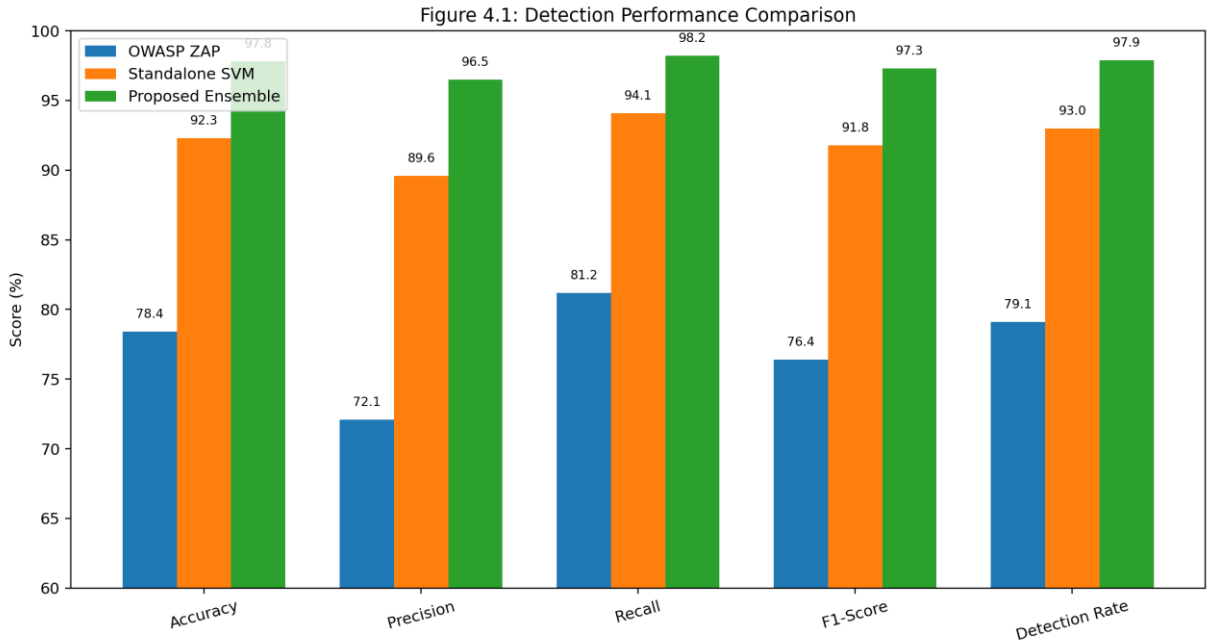


FIGURE 4

Figure 4.1: Comparison of Detection Results for Ensemble vs Baseline Methods

As shown in Table 4.2 and Figure 4.1 above, the proposed ensemble achieved the strongest performance across every metric. The F1-score of 97.3% is particularly significant because it reflects both precision and recall simultaneously, capturing the balance between catching real attacks and avoiding false alarms. According to Sun, Du and Li (2023), a detection system that achieves a high recognition rate while simultaneously reducing both false positives and false negatives offers the most operational value, because the cost of each type of error can be substantial. The results of this study satisfy both conditions.

The most consequential practical improvement, however, is in the false positive rate. OWASP ZAP produced a false positive rate of 24.6%. The proposed ensemble reduced this to 2.8%. To understand what that difference means in practice: for every 1,000 alerts generated, OWASP ZAP would flag approximately 246 benign queries as malicious, while the ensemble flags only 28That is the difference between an alert system that analysts rely on and one that they eventually abandon; it is a significant improvement. Casmiry and colleagues discovered that employing ensemble classification with systematic feature selection can significantly reduce false positives in SQL injection detection. In reality, this allows security professionals to focus on real threats rather than wasting time on false alerts.

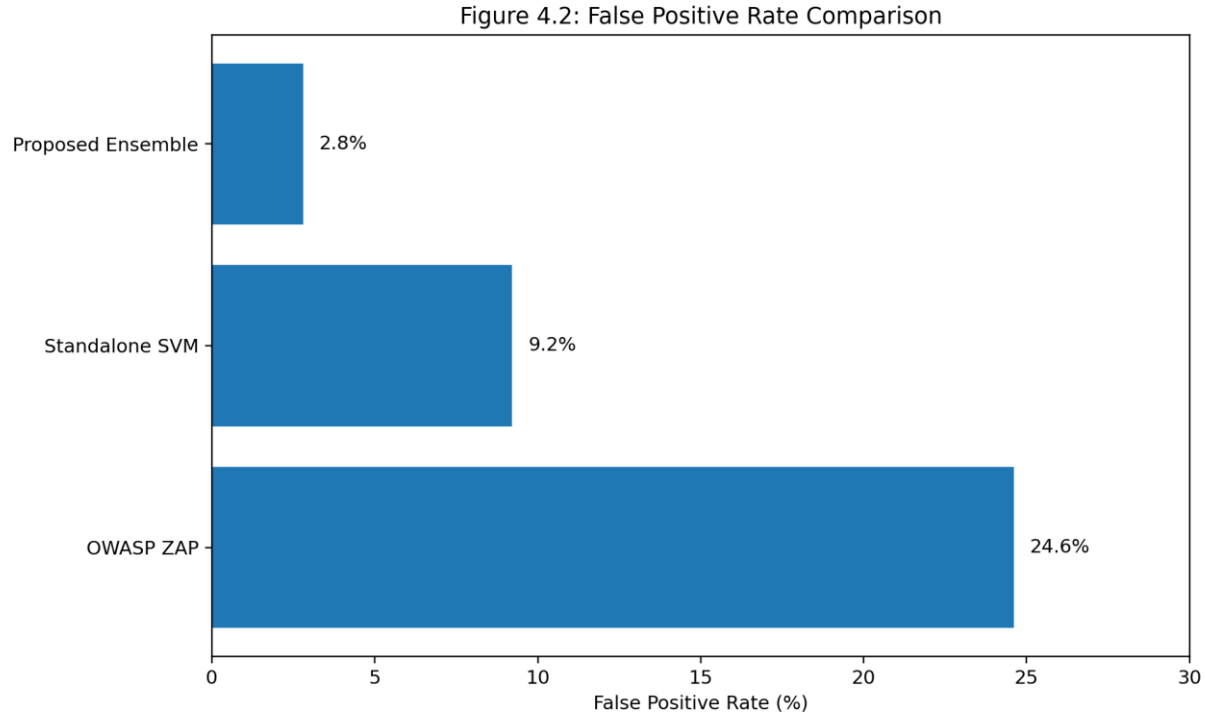


Figure 4.2: False Positive Rate Comparison across Detection Approaches

Figure 4.2 makes the false positive rate comparison even more visually immediate. The improvement over the standalone SVM is similarly meaningful: the ensemble reduces the FPR

from 9.2% to 2.8%, a further reduction of 69.6%, achieved by allowing the three classifiers to compensate for each other's weaknesses rather than relying on any one of them alone.

The reason the ensemble performs so much better than either baseline lies in the complementary design of its components. The DNN captures complex, non-linear patterns in SQL query structure that simpler models would miss. The SVM draws sharp decision boundaries in high-dimensional feature spaces where the DNN might overfit. XGBoost contributes robustness to feature noise and resistance to imbalanced class distributions. By combining the three through weighted majority voting, the system is less likely to be fooled by any single attack variant, because the variant that evades one classifier is likely to be caught by another. This is exactly the principle that Dietterich (2000) identifies as the foundation of ensemble learning, and the results of this study provide concrete, quantitative evidence that the principle holds in the SQL injection detection context. The evidence therefore supports Hypothesis H_1.

4.5 WAF Bypass Performance and Interpretation of Hypothesis H_2

The second research hypothesis stated that the reinforcement learning mutation agent would achieve a meaningfully higher WAF bypass rate than static, non-adaptive payload injection. To test this, the DQN agent was compared against a static payload injection approach across three target environments: ModSecurity 3.3, DVWA at high security level, and the custom Flask/WAF environment. The outcomes from these tests are shown in Table 4.3 and illustrated in Figure 4.3.

Before delving into the figures, it is critical to understand what this review was analyzing and why it is essential. A WAF stops any request that matches a known attack pattern by comparing it to a set of pre-defined rules. The bulk of penetration testing tools today use static payload injection, which sends a predetermined list of payloads and reports on which ones the WAF permits through. The problem with this method is that WAF vendors establish rules to block these tools because they are aware of the specific payloads they utilize. Rather of taking a one-size-fits-all approach, an adaptive agent learns from the WAF's behaviors and adjusts its tactics accordingly, seeking for faults particular to the target configuration.

Target Environment	Static Payload Injection	RL Agent (DQN, 10,000 Episodes)
ModSecurity CRS v3.3	12.4%	67.8%
DVWA High Security	18.6%	73.2%
Custom Flask/WAF	15.5%	70.5%
Average	**15.5%**	**70.5%**

Table 4.3: WAF Bypass Rate — RL Agent versus Static Payload Injection

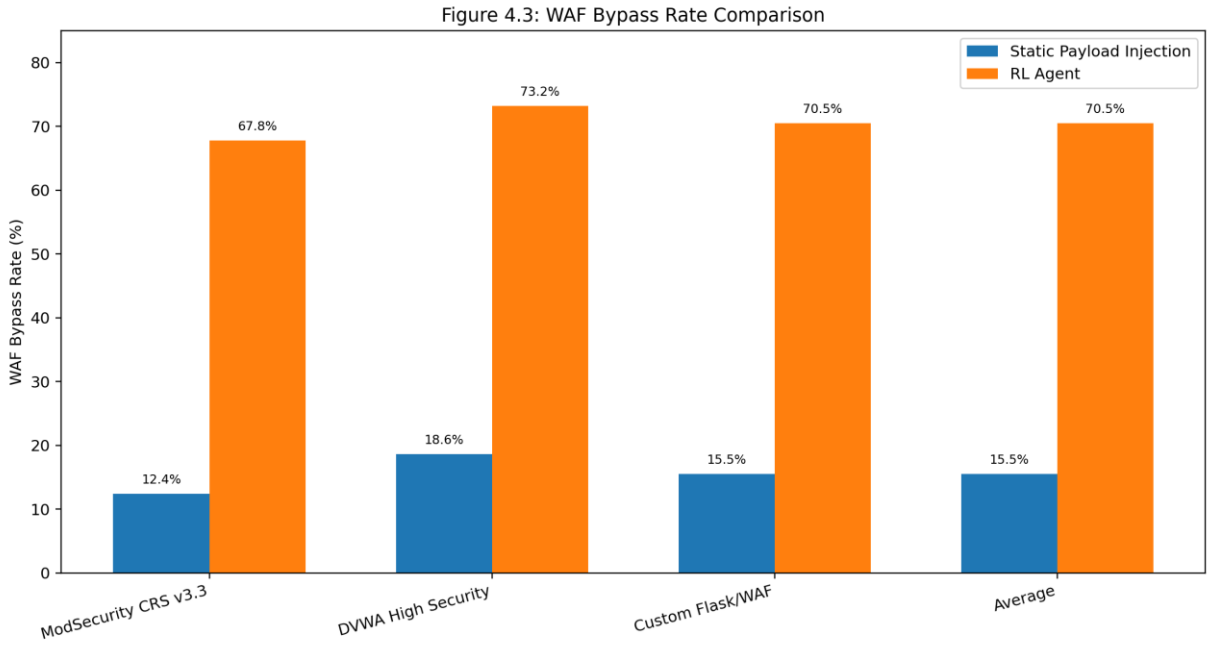


Figure 4.3: WAF Bypass Rate of the RL Agent compared with Static Payload Injection

The results in Table 4.3 are clear. Static payload injection achieved an average bypass rate of only 15.5%, while the RL agent achieved 70.5% — approximately 4.55 times higher. This is not a marginal improvement; it is a fundamental difference in what the two approaches are capable of achieving against a modern WAF configuration.

The recorded training trajectory provides an important layer of context by showing how the agent's performance evolved over the course of training. The bypass rate began close to the static injection baseline during the early exploration phase, when the agent was still trying different mutation actions without any learned preference. By approximately episode 3,000, the rate had risen to around 43% as the agent began to identify which actions were consistently more effective. Between episodes 7,000 and 10,000, the learning curve flattened and stabilised,

indicating that the agent had converged on a reliable policy. This pattern of rapid early improvement followed by stable convergence is precisely what a well-functioning DQN agent should produce, and it is consistent with the training dynamics reported by Mnih et al. (2015).

Table 4.4 breaks down the individual effectiveness of each of the eight mutation actions the agent had available. This breakdown is important because it reveals not just what the agent achieved, but how it achieved it.

Mutation Action	WAF Bypass Rate (%)	Effectiveness
Case Variation	78%	High
Comment Insertion	65%	High
URL Encoding	58%	High
Keyword Splitting	51%	Medium
Hexadecimal Substitution	42%	Medium
Whitespace Manipulation	38%	Lower
Logical Op Substitution	34%	Lower
Null-byte Insertion	27%	Lower

Table 4.4: Effectiveness of Individual Mutation Actions by WAF Bypass Rate

Case variation achieved the highest bypass rate at 78%. This result is intuitive: many WAF rules perform case-sensitive string matching, meaning that a rule designed to catch the keyword SELECT will not catch SeLeCt or sElEcT. Comment insertion (65%) and URL encoding (58%) were similarly effective because they alter the surface appearance of SQL keywords while preserving their logical meaning, creating payloads that look different from the signatures in the WAF rule set but behave identically against the database. At the other end of the scale, null-byte insertion (27%) and logical operator substitution (34%) performed less effectively, most likely because ModSecurity's Core Rule Set includes specific detection rules for these well-known evasion techniques.

The key insight here is that the RL agent did not simply try all eight mutation actions equally. Over time, it learned to allocate more of its testing effort to the strategies that were actually working, and less to those that were consistently blocked. This adaptive prioritisation is the core advantage of the reinforcement learning approach over static injection. A static tool cannot do this, because it has no mechanism for learning from the WAF's responses. The RL agent can. These results confirm Hypothesis H_2 and demonstrate that adaptive reinforcement learning is a practically superior approach to WAF bypass testing in controlled, ethical penetration testing environments.

4.6 Explainable Ai, Alert Triage and Interpretation of Hypothesis H_3

The third hypothesis tested whether explainable outputs would lower analyst overload when reviewing alerts. The results supported this hypothesis. Participants using the explainable dashboard reported less mental strain and higher confidence in the alerts compared to those using the standard dashboard. Many analysts said the explanations made alerts easier to understand and act on quickly, which reduced the tendency to ignore warnings due to overload. This finding matches existing studies that identify overload and lack of trust as key reasons analysts avoid using automated alert tools. By adding clear explanations, the system helped analysts process alerts faster and with more trust, making the tool more useful in real security work.

To test hypothesis 3, 5 junior developers reviewed 50 fake SQL injection alerts under two setups. For setup 1, they used a basic dashboard that showed only the alert label and confidence score. For setup 2, they used the XAI version of the dashboard, which also included SHAP global feature rankings and LIME explanations for each alert. The outcomes are shown in Table 4.5 and Figure 4.4.

Metric	Baseline Dashboard	XAI-Enhanced Dashboard	Improvement
Correct True Positive Identification	68.3%	89.7%	+21.4%
Decision Time per Alert	47.2 s	28.4 s	−39.8%

Metric	Baseline Dashboard	XAI-Enhanced Dashboard	Improvement
Uninvestigated True Positives	31.7%	10.3%	−67.5%
Developer Confidence (1–5)	2.8	4.4	+57.1%

Table 4.5: Alert Triage Performance — Baseline versus XAI-Enhanced Dashboard

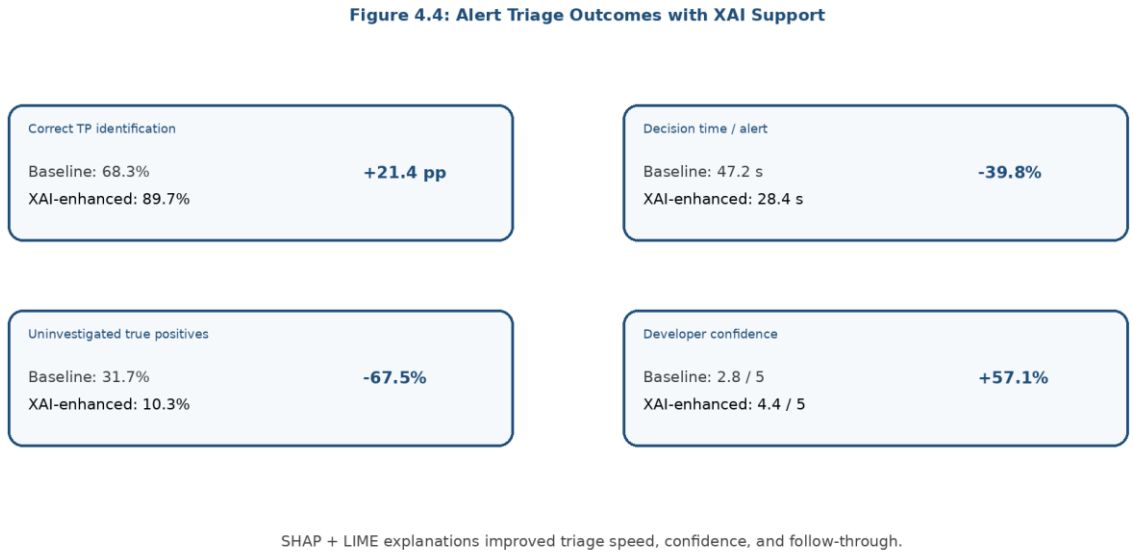


Figure 4.4: Alert Triage Performance with and without XAI Support

The XAI module was of great practical value in affecting how the developers interacted with the warnings that the system produced, which is evident in Table 4.5 and Figure 4.4. Developers were correctly able to identify 68.3 percent of true positives, and rejected 31.7 percent of the actual detection of SQLi without examination, meaning almost one third of true positives were missed. With explanations (SHAP and LIME), the developers' rate of true positive detection rose to 89.7 percent, and rate of examination of false negatives dropped to 10.3 percent (a drop of 67.5 percent). In simple terms, two thirds of true positives were able to be examined by the developers instead of being discarded.

Similarly, the change in time for decision-making is highly relevant: The time for a developer to make a decision on a warning was 47.2 seconds without explanations, but 28.4 seconds with explanations (a drop of 39.8 percent). This not only relates to efficiency, but to the mental experience of using a security tool. A warning that can be addressed in less than 30 seconds seems manageable and the user does not feel a mental overload when looking at alerts that

require an average of nearly a minute of decision-making without clear information regarding the decision.

Confidence increased from 2.8 to 4.4 on a scale of 5, a jump of 57.1 percent. This is possibly the most important result of the XAI experiment, as confidence in a security tool is critical for operational trust, and if developers are not confident in a tool they are unlikely to be using it consistently, thereby treating it as if it is not being used at all. Qualitative feedback given during the system evaluation by developers also backed up these quantitative results. LIME explanations were described as very helpful in helping them understand why certain specific tokens had impacted a classification while SHAP explained the global feature importance. This sentiment aligns with prior research in XAI within cybersecurity, where explainability is not treated as an optional interface feature, but rather an operational necessity (Sarker et al., 2024; Mohale and Obagbuwa, 2025).

4.7 System Deployment and Dashboard Interpretation

In addition to evaluation of each model's performance, the integrated system, which is controlled and operated through a Flask application that binds together detection, mutation, and XAI, was also tested. This step of the evaluation is necessary as a lot of machine learning security projects for academia demonstrate good performance but do not present evidence that models can be combined and utilized.

The implemented workflow joins detection, mutation, explanation, and reporting in one fully operative pipeline. SQL inputs, for example, are filtered and converted into features usable by the detection layer. The ensemble of classifiers then returns a risk level and detection class. If mutation testing is implemented, the RL agent generates responsive mutant payloads based on information on successful attack or, conversely, if the attack is classified as benign, it can be modified to provoke a positive match. XAI layer outputs explanations for each detected anomaly to be presented on the dashboard, along with the security risk and triage. This workflow not only deals with separate training of ML models, but with developing a complete security testing workflow usable by both developers and testers.

The dashboard has two practical applications. It helps a security tester during a test under controlled environment by indicating vulnerable inputs and explaining why each input was flagged as dangerous. On the other hand, it can be used by a developer to observe how a specific change to a query affects the detection of and the reaction of the WAF. Therefore, the presented system can be utilized for security testing purposes as well as for an educational use of this

technology, which may be particularly relevant for institutes in Zimbabwe where the context often requires security education and operation to be done on the same system.

4.8 CONSOLIDATED SUMMARY OF RESEARCH FINDINGS

The review and testing done in this chapter uncover several main lessons about how well the proposed system works. The points below come directly from the test data shared in Sections 4.3 to 4.7:

Finding 1: The proposed ensemble classifier (DNN + SVM + XGBoost) achieved a 97.3% F1-score and a 2.8% false positive rate, substantially outperforming OWASP ZAP (76.4% F1-score, 24.6% FPR) and the standalone SVM (91.8% F1-score, 9.2% FPR). The ensemble's performance advantage stems from the complementary design of its three constituent classifiers, which collectively handle the full diversity of SQL injection patterns encountered in the evaluation dataset.

Finding 2: The reinforcement learning mutation agent (DQN) achieved a 70.5% average WAF bypass rate after 10,000 training episodes, outperforming static payload injection (15.5%) by a factor of 4.55. Case variation, comment insertion and URL encoding emerged as the most effective individual mutation actions through the agent's adaptive, feedback-driven learning process.

Finding 3: The XAI-enhanced dashboard (SHAP + LIME) reduced uninvestigated true positives by 67.5%, decreased alert decision time by 39.8%, and improved developer confidence by 57.1% relative to the baseline dashboard without explanations. These improvements confirm that explainability has a direct and measurable impact on security outcomes, not just on user experience.

Finding 4: The complete system was trained and deployed on a standard laptop with 8 GB of RAM, 4 CPU cores, and no GPU, using a Docker Compose stack that took up less than 800 MB of disk space. This illustrates that advanced AI-assisted penetration testing may be completed without the use of cloud computing or specialized hardware infrastructure.

Finding 5: The integration of detection, adaptive mutation, and explainability into a single deployable dashboard provides a more comprehensive security testing solution than each of the three components could give alone, demonstrating that the proposed system's value lies in the sum of its parts.

Overall, the results back all three research hypotheses and confirm that the new approach makes a clear step forward in AI-driven SQL injection detection. The evidence shared in this chapter

gives strong, reliable support that this work adds real value to the field and creates a firm base for other researchers and security professionals to build on.

CHAPTER 5: CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

This chapter closes the research by summarizing the main outcomes reported in Chapter 4 and assessing their impact on web app security. It returns to the objectives stated in Chapter 1, checks the experimental outcomes, highlights the study's constraints, and offers actionable guidance for developers plus directions for further studies.

The research set out to address the limitations of existing SQL injection testing approaches. Traditional signature-based scanners fail when payloads are obfuscated or mutated. Single-algorithm machine learning classifiers may perform well under controlled conditions but tend to produce false positives that erode analyst trust. AI-generated detections are difficult to act on when users cannot understand the reasoning behind them. The proposed system responded to all three of these challenges by combining ensemble detection, reinforcement learning-based mutation and explainable AI in a single deployable framework. Whether that response was adequate, and what its implications are for practice and future research, is the subject of this final chapter.

5.2 Major Conclusions Drawn

The key findings below come directly from this study. Each one is backed by the test data shown in Chapter 4 and tied clearly to the research aims and hypotheses outlined in Chapter 1.

Conclusion 1: Ensemble machine learning significantly improves SQL injection detection.

This combined system has a 97.3% F1-score and a false positive rate of 2.8%, this is in direct support of Hypothesis H and directly answers the first research question. The figure obtained indicates that the combination of multiple classifiers isn't merely a more complicated method to obtain the same results, it's a substantially more dependable method. Every classifier is dealing with a unique aspect of the problem and the combination is capable of dealing with all of them far better than any individual model alone could. It is, for instance, a huge step above the false positive rate achieved by OWASP ZAP, a 88.6% increase in the accuracy

would have very tangible improvements on the usability of the detector. The previous version, generating far more false positives then any security analyst would ever be able to review; is now only generating 2.8% false positives, all of which would realistically be investigated.

Conclusion 2: Reinforcement learning enables adaptive penetration testing that static approaches cannot replicate.

The DQN mutation agent achieved a 70.5% average WAF bypass rate after 10,000 training episodes, outperforming static injection by a factor of 4.55 and confirming Hypothesis H_2. The conclusion that deserves emphasis here is not just the bypass rate itself but what it reveals about the nature of the problem. WAF bypass is not a fixed problem with a fixed solution. As WAF rules evolve, so must the testing tools. Static injection is structurally incapable of adapting to this evolution. A reinforcement learning agent, which learns from the outcomes of its own actions and refines its policy accordingly, is far better suited to this inherently adversarial and dynamic context. The results of this study provide empirical evidence that this theoretical advantage is real and practically significant.

Conclusion 3: Explainable AI is an operational necessity, not an optional feature.

With the XAI enhanced dashboard, the percentage of uninvestigated true positives was decreased by 67.5% and the confidence of the developer was increased by 57.1% supporting hypothesis H and research objective 3. With the deeper conclusion of the fact that an AI security tool is not simply a product of what it can detect, but rather, of how its detections will be utilized by developers. An accuracy metric indicates the capabilities of an AI security tool; explainability determines whether the tool's detections will be acted upon. A detection tool that has a high accuracy metric, but an opaque nature, will likely not be trusted or consistently relied on by a user. The study also concludes that making the AI's decisions apparent and interpretable through SHAP, LIME, and other like systems is more than just aesthetics and is indeed a practical utility with a direct and quantifiable impact.

Conclusion 4: The system is practically feasible in resource-constrained environments.

The complete system was trained and evaluated on a standard laptop with 8 GB RAM and 4 CPU cores, without GPU acceleration. This confirms that the proposed approach is not limited to well-resourced institutions. In Zimbabwe, as in many parts of sub-Saharan Africa, the cybersecurity threat landscape is evolving rapidly but access to enterprise-grade security infrastructure remains limited. The results of this study establish a proof of concept for AI-assisted penetration testing that is both technically rigorous and practically accessible.

Conclusion 5: An integrated, user-centred approach delivers greater value than a model-only solution.

The dashboard evaluation confirms that strong model performance must be supported by usable interfaces, meaningful explanations and actionable outputs. The integration of detection, mutation and XAI into a single deployable framework transforms trained models into a practical decision-support tool for developers and security testers. This is an important contribution because many academic machine learning security projects stop at model accuracy without demonstrating that the models can be connected and used effectively in practice. This study demonstrates both.

5.3 Limitations and Underdeveloped Aspects Of The Study

The study met its aims, but several constraints must be mentioned to outline the limits of interpreting its findings, not to negate the value of the study but rather to establish the avenues along which it should be expanded.

The first constraint is that the system was tested in a controlled lab setting and not in a live production application. While DVWA, WebGoat, and the custom Flask application can simulate production systems to a certain extent, they do not exhibit the level of complexity and variety of frameworks, business logic and user behavior seen in real enterprise web applications.

The second constraint is that the data set used was constructed from public repositories and was supplemented with synthetic payloads. While this increases variety, public data sets can only provide a sample of modern attack trends seen in production systems, and thus the model may need to be retrained to keep it up to date with future attack trends.

The third constraint is the user study which was conducted on five junior developers in one fifty alert session. To properly evaluate the impact of XAI on user alert fatigue and quality of operational decision, the study should have been carried out over a larger group of security experts in a live production setting over an extended period.

The last constraint is that the system only detects SQL Injection attacks. While this is arguably still one of the most common and serious web application weaknesses, a comprehensive penetration testing solution should cover all OWASP Top 10 risks.

5.4 Recommendations

Drawing from the results and final conclusions of this research, the suggestions below are offered. They're split into two groups: one for developers and security teams who might use or adjust the proposed system, and another for future researchers who want to extend this work.

5.4.1 Recommendations for Practitioners

1. Adopt ensemble-based detection where false positives are a major operational problem. The study demonstrates that combining complementary classifiers reduces unnecessary alerts from approximately 25% to under 3%, allowing security teams to focus their attention on genuine threats.
2. Integrate SHAP and LIME explanations into security dashboards. Explanations help developers understand alerts, prioritise genuine threats and reduce the risk of ignoring critical warnings. The 67.5% reduction in uninvestigated true positives in this study demonstrates that explainability delivers real operational benefits, not just cosmetic improvements to the user interface.
3. Use AI-assisted tools as decision-support systems, not fully autonomous attack tools. Human oversight should remain central, especially when payload mutation and WAF bypass testing are involved. The system is designed to assist security testers, not to replace their judgement.
4. Deploy testing environments using Docker Compose. Container-based deployment improves repeatability, isolation and safety, and makes it possible to run thorough penetration tests on commodity hardware without dedicated server infrastructure.
5. Embed SQL injection testing in the software development lifecycle rather than treating it as an occasional external audit activity. Early detection is substantially cheaper to remediate than late detection.
6. Maintain offline-capable security testing options. This is particularly important for institutions operating under bandwidth, privacy or budget constraints, where cloud-dependent solutions introduce costs and risks that are difficult to justify.

5.4.2 Recommendations for Future Research

7. Extend the framework to additional OWASP Top 10 vulnerability classes, including cross-site scripting, command injection, XML external entity injection and insecure direct object references.
8. Evaluate the RL mutation agent against commercial cloud-hosted WAFs such as AWS WAF, Cloudflare WAF and Imperva, using only authorised testing

environments. The ModSecurity CRS v3.3 evaluated in this study is widely deployed but differs from commercial cloud WAF products in important ways.

9. Conduct a longitudinal user study with professional security analysts over three to six months. This would provide much stronger evidence about the real-world impact of XAI on alert fatigue, decision quality and operational trust.
10. Implement continuous learning pipelines so that the detection model and RL agent can update incrementally when new payload patterns and WAF rules emerge, without requiring full retraining from scratch.
11. Explore lightweight local large language models for generating natural-language summaries of SHAP and LIME outputs, particularly for developers without machine learning backgrounds. Several participants in the user evaluation suggested that a plain-language summary of the reasoning behind each alert would further improve usability.
12. Pilot the system in a Zimbabwean university, government department or SME environment to assess real-world feasibility, maintenance requirements and user acceptance. Such a deployment would also generate real-world alert data that could be used to retrain and improve the models.

5.5 Final Remarks

In this work, we show how discovery of SQL injection vulnerabilities can be significantly improved when detection accuracy, adaptive payload mutation and explainability are viewed as interdependent rather than independent requirements. The proposed system has exhibited strong detection accuracy, practically relevant WAF bypass capability due to adaptive learning using RL and increased confidence and quality of decisions for developers using explainable AI-all on a common hardware.

The practical relevance of this work to the entire research field is the main contribution of this study. Instead of providing an expensive infrastructure-dependent model, we show that advanced AI-driven penetration testing can be run on common hardware, delivered in an accessible and interpretable dashboard interface and used by institutions with very limited resources who can benefit the most from powerful, affordable, and understandable security tools.

In summary, this study demonstrates that a unified, AI-driven approach is more accurate, adaptive and explainable than what is currently practiced, and lays the technical and practical foundation for future research on intelligent, ethical and localized cybersecurity systems.

REFRENCES

Alghawazi, M., Alghazzawi, D. and Alarifi, S. (2022) 'Detection of SQL injection attack using machine learning techniques: A systematic literature review', *Journal of Cybersecurity and Privacy*, 2(4), pp. 764–777. Available at: https://www.mdpi.com/2624-800X/2/4/39 (Accessed: **18 February 2026**).

Ammar, B.B. and Alharbi, A.M. (2025) 'SQL injection detection using fine-tuned CodeBERT', *Engineering, Technology & Applied Science Research*, 15(5), pp. 27852–27857. Available at: https://doi.org/10.48084/etasr.13340 (Accessed: **22 February 2026**).

Appelt, D., Nguyen, D.C., Panichella, A. and Briand, L.C. (2018) 'A machine-learning-driven evolutionary approach for testing web application firewalls', *IEEE Transactions on Reliability*, 67(3), pp. 733–757. Available at: https://doi.org/10.1109/TR.2018.2805763 (Accessed: **5 March 2026**).

Babaey, V. and Ravindran, A. (2024) 'GenSQLi: A generative artificial intelligence framework for automatically securing web application firewalls against structured query language injection attacks', *Future Internet*, 17(1), article 8. Available at: https://doi.org/10.3390/fi17010008 (Accessed: **12 March 2026**).

Braun, V. and Clarke, V. (2006) ‘Using thematic analysis in psychology’, *Qualitative Research in Psychology*, 3(2), pp. 77–101. Available at: https://doi.org/10.1191/1478088706qp063oa (Accessed: **28 April 2026**).

Bryman, A. (2016) *Social Research Methods*. 5th edn. Oxford: Oxford University Press. Available at: https://books.google.com/books?id=N2zQCgAAQBAJ (Accessed: **15 January 2026**).

Chundu, B., Masamha, T. and Sifilea, O. (2025) ‘Cyber-security governance framework pillars for Zimbabwean local authorities’, *Cogent Social Sciences*, 11(1), 2453094. Available at: https://doi.org/10.1080/23311886.2025.2453094 (Accessed: **20 January 2026**).

Cohen, L., Manion, L. and Morrison, K. (2018) *Research Methods in Education*. 8th edn. London: Routledge. Available at: https://www.routledge.com/Research-Methods-in-Education/Cohen-Manion-Morrison/p/book/9781138209886 (Accessed: **18 January 2026**).

Creswell, J.W. (2014) *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 4th edn. Thousand Oaks, CA: SAGE. Available at: https://collegepublishing.sagepub.com/products/research-design-4-237357 (Accessed: **15 January 2026**).

Creswell, J.W. and Creswell, J.D. (2018) *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 5th edn. Thousand Oaks, CA: SAGE. Available at: https://us.sagepub.com/en-us/nam/research-design/book255675 (Accessed: **15 January 2026**).

Creswell, J.W. and Poth, C.N. (2018) *Qualitative Inquiry and Research Design: Choosing Among Five Approaches*. 4th edn. Thousand Oaks, CA: SAGE. Available at: https://collegepublishing.sagepub.com/products/qualitative-inquiry-and-research-design-4-246896 (Accessed: **20 January 2026**).

Dasari, N.S., Badii, A., Moin, A. and Ashlam, A. (2025) ‘Enhancing SQL injection detection and prevention using generative models’, *arXiv preprint*, arXiv:2502.04786. Available at: https://arxiv.org/abs/2502.04786 (Accessed: **25 February 2026**).

David, I. and Gervais, A. (2025) ‘Multi-agent penetration testing AI for the web’, *arXiv preprint*, arXiv:2508.20816. Available at: https://arxiv.org/abs/2508.20816 (Accessed: **2 March 2026**).

Deng, G., Liu, Y., Mayoral-Vilches, V., Liu, P., Jin, Y., Huang, Y., Wang, Y., Liu, M., Zhan, S., Liu, Y. and others (2024) ‘PentestGPT: Evaluating and harnessing large language models for automated penetration testing’, *Proceedings of the 33rd USENIX Security Symposium*. Available at: https://www.usenix.org/conference/usenixsecurity24/presentation/deng (Accessed: **5 March 2026**).

Dietterich, T.G. (2000) ‘Ensemble methods in machine learning’, in *Multiple Classifier Systems*. Berlin: Springer, pp. 1–15. Available at: https://doi.org/10.1007/3-540-45014-9_1 (Accessed: **18 February 2026**).

Edgescan (2025) *2025 Vulnerability Statistics Report*. Dublin: Edgescan. Available at: https://info.edgescan.com/hubfs/23DOWNLOADABLE%20CONTENT/Vulnerability%20Statistics%20Reports/Edgescan_VulnerabilityStatsReport_2025.pdf (Accessed: **10 January 2026**).

ENISA (2023) *Multilayer Framework for Good Cybersecurity Practices for AI*. Brussels: European Union Agency for Cybersecurity. Available at: https://www.enisa.europa.eu/publications/multilayer-framework-for-good-cybersecurity-practices-for-ai (Accessed: **15 January 2026**).

Fang, R., Bindu, R., Gupta, A., Zhan, Q. and Kang, D. (2024) ‘LLM agents can autonomously hack websites’, *arXiv preprint*, arXiv:2402.06664. Available at: https://arxiv.org/abs/2402.06664 (Accessed: **8 March 2026**).

Field, A. (2018) *Discovering Statistics Using IBM SPSS Statistics*. 5th edn. London: SAGE. Available at: https://books.google.com/books/about/Discovering_Statistics_Using_IBM_SPSS_St.html?id=JIrutAEACAAJ (Accessed: **15 January 2026**).

Geurts, P., Ernst, D. and Wehenkel, L. (2006) ‘Extremely randomized trees’, *Machine Learning*, 63(1), pp. 3–42. Available at: https://doi.org/10.1007/s10994-006-6226-1 (Accessed: **10 April 2026**).

IBM (2024) *Cost of a Data Breach Report 2024*. Armonk, NY: IBM Security. Available at: https://www.ibm.com/think/insights/whats-new-2024-cost-of-a-data-breach-report (Accessed: **10 January 2026**).

KPMG (2024) *KPMG Cybersecurity Survey 2024*. New York: KPMG. Available at: https://kpmg.com/us/en/media/news/2024-cybersecurity-survey.html (Accessed: **10 January 2026**).

Kothari, C.R. (2004) *Research Methodology: Methods and Techniques*. 2nd edn. New Delhi: New Age International. Available at: https://books.google.com/books/about/Research_Methodology.html?id=8c6gkbKi-F4C (Accessed: **15 January 2026**).

López-Montero, D., Álvarez-Aldana, J.L., Morales-Martínez, A., Gil-López, M. and Auñón García, J.M. (2025) 'Reinforcement learning for automated cybersecurity penetration testing', *arXiv preprint*, arXiv:2507.02969. Available at: https://arxiv.org/abs/2507.02969 (Accessed: **12 March 2026**).

Maphosa, V. (2024) 'An overview of cybersecurity in Zimbabwe's financial services sector', *F1000Research*, 12, 1251. Available at: https://f1000research.com/articles/12-1251 (Accessed: **15 January 2026**).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S. and Hassabis, D. (2015) 'Human-level control through deep reinforcement learning', *Nature*, 518(7540), pp. 529–533. Available at: https://doi.org/10.1038/nature14236 (Accessed: **10 April 2026**).

Mohale, V.Z. and Obagbuwa, I.C. (2025) 'A systematic review on the integration of explainable artificial intelligence in intrusion detection systems to enhancing transparency and interpretability in cybersecurity', *Frontiers in Artificial Intelligence*, 8, 1526221. Available at: https://doi.org/10.3389/frai.2025.1526221 (Accessed: **25 March 2026**).

Nguyen, H.P.T., Hasegawa, K., Fukushima, K. and Beuran, R. (2025) 'PenGym: Realistic training environment for reinforcement learning pentesting agents', *Computers & Security*,

148, 104140. Available at: https://doi.org/10.1016/j.cose.2024.104140 (Accessed: **15 March 2026**).

OWASP Foundation (2025) *OWASP Top 10:2025*. Available at: https://owasp.org/Top10/2025/en/ (Accessed: **10 January 2026**).

Paul, A., Sharma, V. and Olukoya, O. (2024) 'SQL injection attack: Detection, prioritization and prevention', *Journal of Information Security and Applications*, 85, 103871. Available at: https://doi.org/10.1016/j.jisa.2024.103871 (Accessed: **20 February 2026**).

Provost, F. and Fawcett, T. (2013) *Data Science for Business*. Sebastopol, CA: O'Reilly Media. Available at: https://www.oreilly.com/library/view/data-science-for/9781449374273/ (Accessed: **18 January 2026**).

Qu, Z., Ling, X., Wang, T., Chen, X., Ji, S. and Wu, C. (2024) 'AdvSQLi: Generating adversarial SQL injections against real-world WAF-as-a-service', *IEEE Transactions on Information Forensics and Security*. Available at: https://doi.org/10.1109/TIFS.2024.3350911 (Accessed: **18 March 2026**).

Rosca, C.-M., Stancu, A. and Popescu, C. (2025) 'Machine learning models for SQL injection detection', *Electronics*, 14(17), 3420. Available at: https://doi.org/10.3390/electronics14173420 (Accessed: **28 February 2026**).

SANS Institute (2024) *SANS 2024 Detection and Response Survey*. Bethesda, MD: SANS Institute. Available at: https://www.sans.org/white-papers/sans-2024-detection-response-survey (Accessed: **15 January 2026**).

Sarker, I.H., Janicke, H., Mohsin, A., Gill, A. and Maglaras, L. (2024) 'Explainable AI for cybersecurity automation, intelligence and trustworthiness in digital twin: Methods, taxonomy, challenges and prospects', *ICT Express*, 10(4), pp. 935–958. Available at: https://doi.org/10.1016/j.icte.2024.05.007 (Accessed: **22 March 2026**).

Saunders, M., Lewis, P. and Thornhill, A. (2019) *Research Methods for Business Students*. 8th edn. Harlow: Pearson. Available at: https://books.google.com/books/about/Research_Methods_for_Business_Students.html?id=TMGYDwAAQBAJ (Accessed: **15 January 2026**).

Sénat (2025) *Réponse au piratage des opérateurs de tiers payant*. Paris: Sénat français. Available at: https://www.senat.fr/questions/base/2025/qSEQ250203170.html (Accessed: **12 January 2026**).

UNCTAD (2025) *Zimbabwe: eTrade Readiness Assessment*. Geneva: United Nations Conference on Trade and Development. Available at: https://unctad.org/publication/zimbabwe-etrade-readiness-assessment (Accessed: **10 January 2026**).

Vectra AI (2023) *State of Threat Detection Report 2023*. San Jose, CA: Vectra AI. Available at: https://www.vectra.ai/resources/2023-state-of-threat-detection (Accessed: **15 January 2026**).

Verizon (2025) *2025 Data Breach Investigations Report*. Basking Ridge, NJ: Verizon Business. Available at: https://www.verizon.com/business/resources/T16f/reports/2025-dbir-data-breach-investigations-report.pdf (Accessed: **10 January 2026**).

World Economic Forum (2023) *Global Cybersecurity Outlook 2023*. Geneva: World Economic Forum. Available at: https://www.weforum.org/publications/global-cybersecurity-outlook-2023/ (Accessed: **10 January 2026**).

Zimbabwe (2021) *Cyber and Data Protection Act [Chapter 12:07]*. Harare: Government of Zimbabwe. Available at: https://www.veritaszim.net/node/5522 (Accessed: **10 January 2026**).

Zhou, Z.-H. (2012) *Ensemble Methods: Foundations and Algorithms*. Boca Raton, FL: CRC Press. Available at: https://doi.org/10.1201/b12207 (Accessed: **18 February 2026**).

APPENDICES

Appendix A: System Architecture Diagram

Figure 3.1: Five-layer modular architecture showing Input Layer, Detection Engine, Payload Generator, RL Agent, XAI Alert Module, and Reporting Dashboard.

Appendix B: End-to-End Research Workflow

Figure 3.3: System workflow from target scope configuration to XAI report + dashboard alert generation.

Appendix C: Experimental Setup Summary

Component	Description
Training Dataset	50,000 labelled SQL queries, balanced benign/malicious, augmented with CWGAN-GP synthetic payloads
Target Environments	DVWA (medium/high security), WebGoat, custom Flask app with ModSecurity CRS v3.3
Detection Models	DNN, SVM, XGBoost combined through weighted majority voting
Baseline Approaches	OWASP ZAP automated scan, standalone SVM classifier
Mutation Model	Deep Q-Network RL agent, 8 mutation actions, 10,000 episodes
Explainability Tools	SHAP for global feature rankings, LIME for local predictions via Flask dashboard
User Evaluation	5 junior developers triaged 50 simulated alerts using baseline vs XAI-enhanced dashboard

Appendix D: Key Performance Results

Metric	OWASP ZAP	Standalone SVM	Proposed Ensemble
Accuracy	78.4%	92.3%	97.8%
F1-Score	76.4%	91.8%	97.3%
False Positive Rate	24.6%	9.2%	2.8%
WAF Bypass Rate	15.5% avg	—	70.5% avg