Hybrid Structure Query Language Injection (SQLi) Detection Using Deep Q-Networks: A Reinforcement Machine Learning Model

Carlo Jude P. Abuda[1], Cristina E. Dumdumaya[2]
College of Information and Computing, University of Southeastern Philippines, Davao, City, Philippines[1, 2]
Department of Information Technology, Visayas State University Alangalang, Alangalang, Leyte, Philippines[1]

***Abstract*—Structured Query Language injection (SQLi) remains one of the most pervasive and dangerous threats to web-based systems, capable of compromising databases and bypassing authentication protocols. Despite advancements in machine learning for cybersecurity, many models rely on static detection rules or require extensive labeled datasets, making them less adaptable to evolving threats. Addressing this limitation, the present study aimed to design, implement, and evaluate a Deep Q-Network (DQN) model capable of detecting SQLi attacks using reinforcement learning. The research employed a Design and Development Research (DDR) methodology, supported by an evolutionary prototyping framework, and utilized a dataset of 30,919 labeled SQL queries, balanced between malicious and safe inputs. Preprocessing involved query normalization and vector encoding into fixed-length ASCII representations. The DQN model was trained over 2,000 episodes, using experience replay and an epsilon-greedy strategy. Key evaluation metrics—accuracy, cumulative reward, and epsilon decay—showed performance improvements, with accuracy increasing from 52% to 82% and stabilizing between 65% and 73% in later episodes. The agent demonstrated consistent adaptability by successfully generalizing across various injection patterns. This outcome suggests that reinforcement learning, particularly using DQN, provides a viable alternative to traditional models, with superior resilience and dynamic learning capabilities. The model's convergence trend highlights its practical application in real-time SQLi detection systems, contributing significantly to cybersecurity measures for database-driven applications.**

***Keywords*—Adaptive systems; cybersecurity; deep q-network; intrusion detection; query classification; reinforcement learning; SQL injection**

I. Introduction

Structured Query Language Injection (SQLi) is a malicious technique that enables attackers to interfere with the queries that an application makes to its database [1]. As statistics shows, this remains one of the most critical threats in cybersecurity [2], frequently exploited to bypass authentication [3], retrieve confidential data [4], or even manipulate databases [5]. Understanding the core types of SQLi is essential in developing effective countermeasures. Starting with In-band SQLi (also known as classic SQLi) allows attackers to use the same communication channel for both launching the attack and gathering results [6]. Inferential SQLi, or blind SQLi, enables attackers to reconstruct the database structure based on application behavior and response time without direct data retrieval [2]. Out-of-band SQLi, meanwhile, leverages separate channels such as Domain Name System (DNS) or Hypertext Transfer Protocol (HTTP) requests to exfiltrate data, often when direct feedback mechanisms are disabled [3].

The persistent nature of SQLi attacks underlines the importance of continuous innovation in threat detection. Traditional approaches like signature-based detection [7] and rule-based filtering [8] often fail to keep up with new attack variants. More recently, anomaly detection models and deep learning algorithms, including Long Short-Term Memory (LSTM) networks, have been deployed to detect suspicious patterns in SQL queries [9]. Despite their success, these models face significant drawbacks such as overfitting, high false positive rates, and challenges in recognizing sophisticated or obfuscated attack vectors [10].

Several machine learning (ML) algorithms—such as decision trees, support vector machines (SVMs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs)—have demonstrated promising results in identifying SQLi behaviors [11][12]. However, they still struggle with issues like computational complexity and a lack of adaptability to evolving threats [13]. A notable limitation of these models is their dependence on large labeled datasets and static learning paradigms, which reduce their effectiveness in dynamic environments.

To address these limitations, existing research have begun exploring the capabilities of Reinforcement Learning (RL), a model-free learning paradigm where agents learn optimal actions through interaction with their environment [14]. In particular, Deep Q-Networks (DQNs) combine Q-learning with deep neural networks to approximate action-value functions and make intelligent decisions [15]. Moreover, DQN can adjust detection strategies based on feedback, which makes them suitable for dynamic, real-time security scenarios [16]. Enhancements in reward function design, policy optimization, and experience replay mechanisms have enabled DQNs to outperform conventional models in several intrusion detection use cases [17].

However, two significant research gaps have emerged. First, most existing DQN-based intrusion detection studies do not focus exclusively on SQLi detection using diverse query

datasets [14]; and second, many models are trained and tested using synthetic or simplified datasets that do not accurately reflect real-world injection techniques. And based on researchers it was reported with a high accuracy rates for anomaly detection models but acknowledged that their dataset lacked common obfuscation and encoding schemes found in actual attacks [18][19].

There is a noticeable gap in research dedicated to the application of reinforcement learning in SQLi prevention [20][21], local studies have emphasize static defense mechanisms like input validation or firewall implementation. For instance, a research was conducted [22] to study on common SQLi attack vectors in the e-commerce platforms but proposed only conventional validation techniques as countermeasure.

As drawbacks were evidently presented regarding the various gaps among existing models, this research seeks to address the gaps by developing a DQN-based model specifically designed to detect SQLi attacks. Furthermore, the specific objectives are to preprocess SQL queries into state representations suitable for reinforcement learning; design and implement a DQN model for SQLi detection; and evaluate the model's accuracy, adaptability, and performance across multiple episodes using labeled datasets. Additionally, the aim of this research is also to contribute a hybrid, dynamic, and intelligent framework for mitigating SQLi attacks in web-based systems, thus integrating reinforcement learning into cybersecurity applications.

Additionally, this study also aimed to contribute to Sustainable Development Goal (SDG) No. 9: Industry, Innovation, and Infrastructure, which emphasizes the advancement of reliable, sustainable, and resilient digital infrastructure through scientific innovation. By introducing a Deep Q-Network-based detection model against SQL injection attacks, the research promotes the integration of cutting-edge cybersecurity mechanisms into web systems. Strengthening the security foundations of digital platforms not only supports industrial innovation but also enhances trust in digital technologies that is an essential element in building inclusive and secure infrastructures in today's interconnected society.

However, the scope of this study is limited to the development and evaluation of the model itself and does not extend to the creation of a user interface or the full deployment pipeline for applying the model in production environments. Moreover, this study does not cover the identification and classification of specific query structures such as subqueries, inner queries whether independent or correlated, scalar queries, column queries, row queries, or table queries. The focus remains solely on detecting the presence of SQLi patterns at the query level without dissecting or categorizing the internal query composition.

II. Review of Related Studies

A. Feasbility of Reinforcement Machine Leaning Model

Preprocessing SQL queries is a critical step in developing machine learning models for SQLi detection. This process involves transforming raw SQL queries into structured formats that can be effectively analyzed by machine learning algorithms. The primary goal is to convert the unstructured text of SQL queries into numerical representations that capture the essential features of the queries while preserving their semantic meaning. One fundamental technique in preprocessing is tokenization, which involves breaking down a SQL query into its constituent components, such as keywords, operators, and operands [23]. This segmentation facilitates the identification of patterns and anomalies within the queries. For instance, in the SQL query SELECT * FROM users WHERE username = 'admin' AND password = 'password', tokenization would separate the query into individual elements like SELECT, *, FROM, users, WHERE, username, =, 'admin', AND, password, =, and 'password'. By analyzing these tokens, machine learning models can more easily detect unusual or malicious patterns indicative of SQLi attempts [24].

Beyond tokenization, parsing is employed to understand the syntactic and semantic relationships between the tokens [25]. Parsing involves analyzing the grammatical structure of the SQL query to build a parse tree or abstract syntax tree that represents the hierarchical relationships between different components of the query [26]. This structured representation allows for a deeper understanding of the query's intent and can help in identifying complex injection patterns that simple token-based analysis might miss.

After tokenization and parsing, the next step is vectorization, where the structured representations are converted into numerical formats suitable for input into machine learning algorithms [27]. One common approach is to use techniques like word embeddings, where each token is mapped to a high-dimensional vector that captures its semantic meaning. Methods such as Word2Vec or GloVe [28] can be employed to generate these embeddings, allowing the model to understand similarities and relationships between different tokens based on their contextual usage in a large corpus of text. An alternative vectorization method involves creating bag-of-words (BoW) or term frequency-inverse document frequency (TF-IDF) representations [29]. In these approaches, each query is represented as a vector of token frequencies, either as raw counts (BoW) or weighted by the inverse frequency of the token across the entire dataset (TF-IDF) [30][31]. While these methods are simpler and less computationally intensive than word embeddings, they may not capture the semantic relationships between tokens as effectively.

The choice of preprocessing techniques can significantly impact the performance of the SQLi detection model. For example, Santos et al. [32] proposed a method that involves analyzing SQL queries by stripping parameters to form generalized query structures, enabling the detection of structural deviations indicative of potential attacks [33]. Their approach demonstrated that by focusing on the structural aspects of SQL queries, it is possible to identify anomalies that may signify injection attempts.

Similarly, Shah et al. (2022) developed a deep neural network-based detection model that converts SQL data into word vectors, forming a sparse matrix input for training. Their model incorporated multiple hidden layers with Rectified Learning Unit (ReLU) activation functions and optimized loss functions, achieving an accuracy exceeding 76%. This study

highlights the effectiveness of using deep learning architectures in conjunction with advanced preprocessing techniques to capture complex patterns associated with SQLi attacks [34].

Effective preprocessing also involves handling noise and irrelevant information in the SQL queries [32]. This may include removing comments, extra whitespace, or other non-essential elements that do not contribute to the semantic meaning of the query but could introduce variability that confounds the model [34]. By cleaning the queries and standardizing their format, the model can focus on the meaningful components that are indicative of normal or malicious behavior.

Another important consideration is the handling of dynamic elements within SQL queries, such as user inputs or session variables [35]. These elements can introduce variability and complexity into the queries, making it more challenging to detect injections. Techniques such as parameterization or the use of placeholders can help in normalizing these dynamic components, allowing the model to focus on the structural patterns of the queries [36]. Furthermore, the preprocessing pipeline should be designed to handle multilingual or locale-specific elements, especially in applications that support multiple languages or character sets. Ensuring that the tokenization and parsing processes are robust to different languages and encodings is crucial for maintaining the effectiveness of the SQLi detection model across diverse user bases. Incorporating contextual information into the preprocessing stage can also enhance the model's performance [37]. This may involve considering the source of the query, the role of the user executing it, or the application's state at the time of the query. By integrating this contextual data, the model can make more informed decisions about the likelihood of a query being malicious.

Moreover, the preprocessing techniques should be evaluated for their computational efficiency, especially in real-time detection scenarios. Techniques that are too computationally intensive may introduce latency, which is unacceptable in high-performance applications. Balancing the depth of analysis with the need for speed is a key consideration in the design of the preprocessing pipeline [38]. Finally, it is essential to continuously update and refine the preprocessing techniques to adapt to evolving SQLi tactics. Attackers continually develop new methods to evade detection, and the preprocessing pipeline must be agile enough to incorporate new patterns and anomalies as they emerge. Regularly updating the tokenization, parsing, and vectorization methods, as well as retraining the detection models with recent data, can help maintain the effectiveness of the SQLi detection system [39].

In summary, preprocessing SQL queries into state representations suitable for reinforcement learning involves a series of steps aimed at transforming raw queries into structured, numerical formats that capture their semantic essence. Techniques such as tokenization, parsing, and vectorization are employed to break down queries into their fundamental components, understand their structural relationships, and convert them into formats amenable to machine learning analysis. Effective preprocessing enhances the model's ability to detect anomalies and improves the overall accuracy of SQLi.

B. Existing Methods Integrated with DQN

Designing and implementing a DQN model for web vulnerability detection combines concepts from both deep learning and reinforcement learning to provide a dynamic and intelligent solution to one of the most persistent threats in web security [40]. A DQN is a reinforcement learning algorithm that uses a neural network to approximate Q-values, which represent the expected rewards of taking certain actions in specific states. Unlike traditional machine learning models that require manually labeled input-output pairs, reinforcement learning models like DQN learn through interaction with an environment. In the context of SQLi detection, this "environment" can be simulated using a dataset of labeled SQL queries, including both legitimate and malicious examples [41].

The model learns by receiving feedback when it correctly identifies an injection attack, it receives a positive reward; when it fails, it receives a penalty. Over time, the agent becomes more accurate in identifying which features of SQL queries indicate an attack [42]. A key part of this process is defining the state space, which involves transforming raw SQL queries into numerical formats that preserve both structure and semantics. These could include vectorized tokens, embeddings, or one-hot encodings based on preprocessed query components. This numerical input is then fed into the DQN's input layer [43]. The architecture typically consists of multiple dense (fully connected) hidden layers, often using ReLU as the activation function, to process and learn patterns in the data.

The output layer of the network contains Q-values representing possible actions the model can take — in this case, labeling a query as either normal or malicious. During training, the DQN updates its internal weights to maximize the total expected reward across all episodes. It uses algorithms like experience replay, which stores past experiences in a memory buffer and samples them randomly during training to break the correlation between sequential data. Another technique used is the target network [44], a separate copy of the Q-network that is updated less frequently to improve stability in learning.

One of the strengths of using DQNs for this problem is adaptability [45]. Unlike static detection systems that rely on fixed rules or signatures, a reinforcement learning model can continuously improve by learning from new attack patterns. It can generalize from past experiences to detect previously unseen types of SQLi attacks, making it especially effective in environments where threats evolve rapidly. Moreover, the model's ability to self-learn reduces the need for continuous human intervention, streamlining the cybersecurity workflow. Researcher from Salah et al. [46] have shown promising results using deep learning models for SQLi detection, achieving high accuracy by allowing the model to learn complex patterns directly from data.

Designing the reward function is a crucial part of the implementation process. It must encourage correct classification while penalizing false positives and false negatives appropriately [47]. A poorly designed reward function could lead the agent to adopt suboptimal policies. Additionally, balancing the exploration and exploitation trade-off is vital: the agent must try new actions to discover better policies (exploration) while using known strategies to maximize reward

(exploitation). This is typically managed using an epsilon-greedy strategy, where the model explores randomly with probability ε and exploits the best-known action otherwise [48].

The success of the DQN model also depends on the quality and diversity of the training data. The dataset must include a wide variety of SQL queries — including obfuscated, encoded, or uncommon attack patterns — to ensure the model learns to detect a broad range of malicious behaviors. In practice, developers may use benchmark datasets or simulate realistic web traffic that includes injection attempts. Once trained, the model must be evaluated using metrics such as accuracy, precision, recall, F1-score, and Area Under the Curve (AUC) to determine its effectiveness. Performance across these metrics helps identify whether the model favors false positives (flagging good queries as attacks) or false negatives (failing to detect actual attacks), both of which have serious implications [49].

Another consideration during implementation is computational efficiency. DQNs require substantial resources to train, especially when using large datasets or deep architectures. This means selecting an appropriate model complexity that balances detection performance with processing speed, especially if the model is to be deployed in real-time environments [50]. Moreover, to avoid overfitting — where the model performs well on training data but poorly on unseen queries — techniques such as dropout, regularization, and cross-validation may be applied. Once trained, the model can be integrated into a web application's backend or a security monitoring system to intercept and evaluate SQL queries in real-time.

Another factor, in the implementation process of this model involves managing the progression of learning phases to maximize training effectiveness. During the early exploration phase, the model is intentionally encouraged to sample a wide range of state-action pairs, typically by employing strategies such as epsilon-greedy exploration [51]. This ensures that DQN does not prematurely converge on suboptimal policies by relying solely on immediate rewards but instead develops a broader understanding of the environment's dynamics. Early exploration is vital in avoiding bias in action selection, particularly when the initial model weights are random and uninformed.

As training proceeds, learning growth becomes evident through the gradual refinement of the Q-function approximation. The model's predictions for future rewards become more accurate, and learning curves typically exhibit a consistent reduction in loss metrics [52]. At this stage, computational efficiency techniques such as prioritized experience replay, and target network stabilization are often applied to further optimize the training process without sacrificing generalization.

Eventually, the model enters a phase of policy exploitation, where it leverages its accumulated knowledge to consistently select actions that maximize long-term rewards. Fine-tuning of hyperparameters, including the reduction of exploration rates and adaptive learning rate adjustments, supports this transition from exploration to exploitation [53]. Towards the progression of this process, careful monitoring was observed by the researcher during this phase as this is necessary to prevent overfitting, as the model might otherwise memorize specific patterns in the training dataset, reducing its capacity to generalize to novel queries.

Finally, the training process aims for final convergence, where Q-value estimates stabilize, and policy updates produce negligible changes. Achieving convergence indicates that the DQN has sufficiently learned to distinguish between benign and malicious SQL queries under diverse input conditions. Validation against independent test sets and cross-validation strategies are crucial during this stage to confirm that the model's performance is not limited to training data alone but extends effectively to unseen inputs. Once final convergence is validated, the DQN model can be confidently deployed into a web application's backend or integrated within a real-time security monitoring infrastructure [54].

In summary, designing and implementing a DQN model for SQLi detection involves more than coding a neural network — it requires careful planning, data preparation, environmental simulation, algorithmic tuning, and continuous validation. The strength of this approach lies in its ability to self-learn, adapt, and generalize across a wide range of attack types, making it a promising solution in the ever-evolving field of cybersecurity. By mimicking the behavior of intelligent agents that learn from trial and error, the model contributes not only to improved threat detection but also to building smarter, more secure digital systems.

C. Evaluating the Model's Accuracy, Adaptability, and Performance Across Multiple Episodes Using Labeled Datasets

Evaluating a Deep Q-Network (DQN) model for SQLi detection is a crucial phase in determining its practical value and effectiveness in real-world cybersecurity applications. The evaluation process helps to measure not only how accurately the model detects malicious SQL queries but also how adaptable it is to unseen threats and how consistently it performs across different training and testing episodes. In particular, the parameters includes the following key aspects of 1) accuracy, 2) reward, 3) epsilon decay, and 4) performance stability, these objectively ensures that the model is not just theoretically sound but also operationally reliable when deployed in actual web systems [49].

Accuracy is one of the most fundamental metrics used in evaluating machine learning models. In SQLi detection, accuracy refers to the proportion of correct predictions (both malicious and benign) over the total number of predictions. A high accuracy rate indicates that the model can reliably distinguish between safe and unsafe SQL queries. However, accuracy alone can be misleading, especially when dealing with imbalanced datasets where benign queries significantly outnumber malicious ones. In such cases, other metrics such as precision, recall, and F1-score are more informative. Precision measures how many of the queries flagged as SQLi were actually malicious, while recall determines how many of the actual SQLi queries were successfully identified [55][56]. Then, F1-score is the harmonic mean of precision and recall, offering a balanced measure that is particularly useful when false positives and false negatives carry significant risks.

Beyond these standard metrics, model adaptability is another key dimension to assess. Adaptability refers to the model's ability to maintain performance when exposed to new or previously unseen types of SQL injection attacks . A good DQN model should not just memorize patterns from the training data—it should generalize, learning underlying principles that allow it to detect variants of attacks that were not explicitly present during training. This is especially important in cybersecurity, where attackers frequently change tactics to evade detection [57]. Therefore, part of the evaluation involves exposing the trained model to new datasets or adversarial examples that simulate evolving attack methods and monitoring how well the model maintains its detection capabilities.

Another critical aspect of the evaluation process is observing the model's performance across multiple episodes. In reinforcement learning, the agent interacts with the environment over episodes, learning incrementally based on the rewards received for its actions. Evaluating the model over many episodes ensures that its learning is stable and that performance improvements are not just the result of random fluctuations or overfitting [58]. Performance can be tracked using cumulative reward plots, convergence rates, and episode-wise accuracy metrics. These indicators help identify whether the model is learning effectively or if it is plateauing or regressing in its performance over time.

The quality and diversity of the dataset used for evaluation also play a crucial role. Using a labeled dataset means that every SQL query has been previously classified as either safe or malicious. This allows for objective measurement of the model's predictions. A good evaluation dataset should include a wide range of SQL queries: traditional injection patterns, obfuscated payloads, encoded strings, and even polymorphic SQL attacks [58]. Inclusion of noise and real-world queries that closely mimic normal user behavior adds further robustness to the testing process [59].

To ensure fairness and reproducibility, the evaluation should use standard data splitting techniques. Typically, datasets are divided into training, validation, and test sets. The model is trained on the training set, tuned on the validation set, and its final performance is reported on the test set. Cross-validation techniques, such as k-fold validation [60], can further improve reliability by averaging performance over multiple data partitions [61]. This reduces bias and helps in understanding how the model behaves under different data distributions.

Performance should also be measured in terms of computational efficiency. In practical deployments, a model must make decisions in real-time or near real-time. This means latency—how long it takes to analyze and classify a single query—becomes a critical metric. A high-performing model that takes several seconds to respond may not be suitable for real-time applications such as intrusion prevention systems. Therefore, evaluating the DQN model's inference time, memory consumption, and Central Processing Unit (CPU)/Graphic Processing Unit (GPU) utilization becomes essential, particularly when planning for integration into existing web architectures [62].

Robustness testing is another valuable part of evaluation. This involves intentionally introducing noise, incorrect data formatting, or adversarial inputs to observe whether the model can still make correct classifications [63]. A vigorous SQLi detection model must to not break down or perform erratically when encountering slight deviations from expected input. Testing under these conditions gives insights into the model's stability and readiness for deployment in unpredictable environments.

Furthermore, comparative evaluation against baseline models is vital. The DQN model's performance should be compared with traditional classifiers such as Decision Trees, Support Vector Machines (SVM) [64], Random Forests [65], or even static rule-based systems [66]. If the DQN model consistently outperforms these alternatives across all evaluation metrics, it justifies the additional complexity and computational cost involved in implementing reinforcement learning. Studies such as those by Anwar (2023) [67] and Alghawazi et al. (2023) [68] have demonstrated how deep learning models can significantly surpass conventional techniques in detecting SQLi attacks, particularly in adapting to real-world query patterns and minimizing false alarms.

III. METHODOLOGY

This study employed the Design and Development Research (DDR) methodology approach [69][70] to develop a DQN-based detection model for SQLi attacks. DDR is a research methodology that focuses on designing, building, and evaluating models to solve identified problems—in this research, that the persistent threat of SQL injection in database-driven web systems.

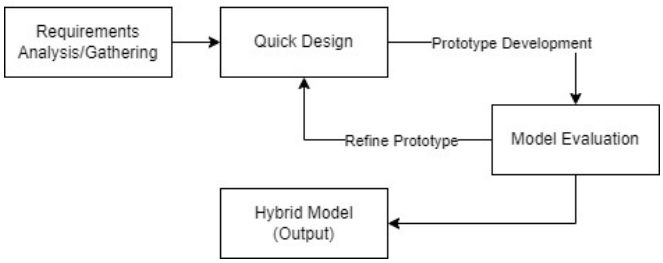

Fig. 1. Research implementation of evolutionary prototype software development life cycle framework.

In Fig. 1, the study further adopted the Evolutionary Prototyping Model [71]–[73] as the Software Development Life Cycle (SDLC) framework, wherein in this approach supported iterative development and refinement of a functional prototype was developed by the researcher to align the reinforcement learning structure of the proposed model.

A. Requirements Analysis and Gathering

The development process began with the requirements analysis phase, during which the nature of SQLi attacks was studied in detail as presented in Pseudocode 1.

Pseudocode 1: RL Development

```
START
    LOAD and preprocess dataset
        NORMALIZE and CLEAN SQL queries
        ENCODE queries into fixed-length vectors
        LABEL each query as Safe or Malicious
    SPLIT data into training and testing sets (80/20)

    DEFINE environment to:
        PROVIDE query input
        RETURN reward based on prediction accuracy
    INITIALIZE DQN agent:
        BUILD neural network
        SET learning parameters (epsilon, gamma, learning rate)

    FOR each training episode:
        RESET environment and GET initial state
        FOR
            SELECT action (predict Safe or SQLi)
            GET reward and next state
            STORE experience
        UPDATE model from memory (experience replay)
        DECAY exploration rate (epsilon)

    PLOT accuracy and reward trends over episodes
END
```

This included a review of known SQLi patterns and classification techniques, which collected from existing or secondary data sets creating a labeled dataset online. The dataset identified both safe and malicious SQL queries, representing various types of attacks such as In-band SQLi, Inferential SQLi, and Out-of-band SQLi. At this stage, the specific need to transform human-readable queries into machine-processable formats was identified, directly addressing the first research objective: to preprocess SQL queries into state representations suitable for reinforcement learning. Next is the quick design phase, it is now the preliminary logic was implemented to preprocess the queries. A custom text normalization function was applied which involved converting all characters to lowercase and removing special characters. Then, to enrich the dataset, a rule-based classifier was also applied to detect and label different SQLi attack types based on keyword patterns. Each query was then encoded into a fixed-length numeric vector using character-level ordinal encoding. This transformation enabled uniform input for the DQN model while preserving critical structural features of the SQL statements.

B. Quick Design, Prototype Development and Refinement

The subsequent phase focused on prototype development, where the actual Deep Q-Network was implemented using Python and TensorFlow. A simulated environment was developed using a custom class SQLiEnv that allowed the agent to interact with the dataset by analyzing queries one at a time. The DQN agent was structured with neural network architecture comprising an input layer, two hidden layers using ReLU activation functions, a dropout layer for regularization, and an output layer with softmax activation for classification. The DQN model was trained to classify each query as either safe or malicious, thus delivering the second research objective of the study that is to design and implement a DQN model for SQLi detection.

In the testing and refinement stage, the developed prototype took place over 2,000 training episodes, each consisting of 100 interactions between the agent and the environment. For each interaction, the model is expected to receive a reward of +1 for correct predictions and -1 for incorrect ones as provided in Fig. 2.

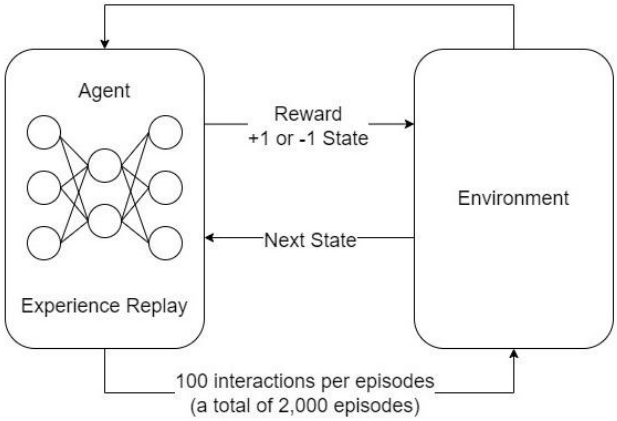

Fig. 2. Reinforcement learning training loop for SQLi detection.

Fig. 2 shows that feedback loops were used to adjust the model's policy over time. The model employed reinforcement learning principles such as experience replay and epsilon-greedy exploration to balance learning from past experiences with discovering new strategies.

In the evaluation of the DQN-based SQLi hybrid detection model was guided by key reinforcement learning metrics. First, accuracy per episode was tracked and stored in the accuracy list. This metric was calculated as the percentage of correct predictions out of 100 interactions per episode, providing a clear measure of how the model's classification ability improved over time. Additionally, the total reward per episode, recorded in the rewards list, reflected how many actions (classifications) were correctly taken by the agent in each training cycle. Since a correct classification yielded a reward of +1 and an incorrect one a reward of -1, the total reward served as a direct indicator of learning success.

1) Model evaluation: To guide the learning behavior, the model employed an epsilon-greedy exploration strategy. The epsilon value, initialized at 1.0, decayed exponentially by a factor of 0.995 after each episode until reaching a minimum of

0.01. This ensured a balance between exploration (trying new actions) and exploitation (using the best-known policy), allowing the agent to learn optimally over time. As training progressed, convergence and stability were observed around among episodes, as indicated by the flattening trend in both accuracy and reward outcomes as expected.

2) Hybrid model (output): The stable metrics referencing from the literatures [74] shows that the agent had learned a near-optimal classification policy and ceased making significant changes in behavior. Furthermore, a learning curve visualization was generated using Matplotlib, which illustrated the progression of both total rewards and classification accuracy over 2,000 training episodes. These evaluation metrics, drawn directly from the model's training logs and source code implementation, demonstrate that the agent not only learned effectively but also maintained consistent performance in SQLi detection tasks.

Moreover, the application and integration of various tools in the study included Python for development, TensorFlow for deep learning modeling, Pandas and NumPy for data handling, Scikit-learn for data partitioning, and Matplotlib for visualization. This toolchain enabled smooth development and evaluation of the prototype in alignment with the DDR methodology and the evolutionary prototyping model.

3) Ethical considerations: The study adhered to ethical research standards by exclusively utilizing secondary datasets sourced from Kaggle's publicly accessible SQL injection repositories. These datasets, contributed for academic and educational purposes, were fully anonymized and contained no personally identifiable information, ensuring that data privacy and confidentiality were consistently protected. Throughout the research process, the researcher complied with Kaggle's licensing terms by restricting the use of the datasets strictly for academic analysis without any redistribution or unauthorized modification. Since the investigation involved no direct engagement with human subjects, institutional review board (IRB) approval and informed consent requirements were deemed unnecessary. The study-maintained transparency, integrity, and responsible data handling practices, aiming to contribute meaningfully to cybersecurity research while upholding the rights and intentions of the original data contributors.

IV. Results and Discussion

The dataset used in this study consisted of a total of 30,919 SQL queries, comprising both malicious and safe inputs. Specifically, 11,382 queries (36.8%) were labeled as SQL injection attacks, while 19,537 queries (63.2%) were labeled as safe queries. This balance provided the DQN agent with a realistic and diverse set of inputs for training and evaluation.

As observed in Table I, it presents sample SQL queries after preprocessing, along with their assigned labels and identified SQLi types. Each raw query was normalized to remove special characters and standardize structure, enabling uniform encoding. The table shows that typical SQL injection patterns—such as 'admin' OR 1=1--—were correctly categorized as In-band SQLi (Classic), while safe queries like SELECT password FROM users were labeled as Unknown/Normal Query. This structured labeling allowed the model to differentiate malicious input from benign ones during training.

Now, for the Table II, the researcher then summarizes the classification output of SQLi types after preprocessing and labeling. Out of the total 30,919 SQL queries, the majority (27,425 or 88.7%) were categorized as Unknown/Normal Queries, while 3,494 queries (11.3%) were identified as In-band SQLi (Classic). No samples were labeled under Inferential SQLi (Blind), reflecting the specific distribution present in the dataset used. This breakdown provided the model with a representative dataset for distinguishing between malicious and safe query types.

TABLE I. Dataset After Preprocessing and SQLi Type Classification

Query (Raw)	Processed Query	Label	SQLi Type
SELECT * FROM users ...	select from users	1	In-band SQLi (Classic)
admin' OR 1=1 --	admin or 11	1	In-band SQLi (Classic)
SELECT password FROM users	select password from users	0	Unknown/Normal Query

TABLE II. SQLi Type Classification

SQLi Type	Count	Percentage (%)
Unknown/Normal Query	27,425	88.7
In-band SQLi (Classic)	3,494	11.30
Inferential SQLi (Blind)	0	0.00

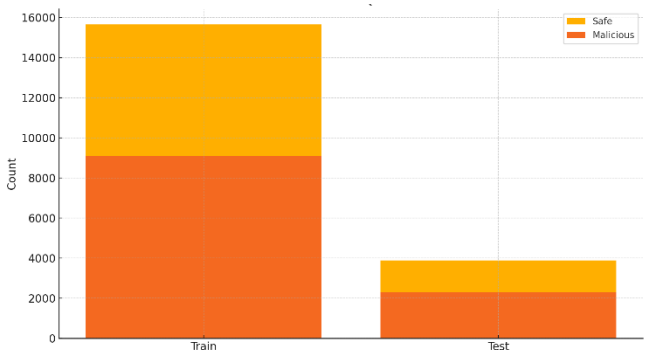

Fig. 3. Distribution of Safe Vs Malicious in train and test sets.

Consequently, Fig. 3 illustrates the distribution of safe versus malicious SQL queries across the training and test sets after applying an 80/20 split. The training set consisted of 24,735 queries, while the test set included 6,184 queries. Both sets preserved the original ratio of benign to malicious inputs, ensuring that the model was exposed to a balanced representation during learning and evaluation phases.

Furthermore it is also significantly observed that the model was capable of learning how to detect SQL injection (SQLi) attacks based on query patterns. Notably, this was proven that a successful integration of a reinforcement learning environment

(SQLiEnv) and a learning agent (DQN Agent) within a simulation loop running across 2,000 episodes. The model was constructed using a neural architecture with an input layer of 100 units (matching the encoded vector length of preprocessed queries), followed by two hidden layers with 128 and 64 neurons activated by ReLU, and a softmax-activated output layer for binary classification (safe vs. SQLi). A dropout layer with a rate of 0.3 was introduced to prevent overfitting, and the model was compiled using categorical cross-entropy loss with the Adaptive Moment Estimation (ADAM) optimizer set at a learning rate of 0.001.

Following the preprocessing and encoding of SQL queries as discussed, the second phase of this study aimed to design and implement a Deep Q-Network (DQN) capable of classifying SQL injection (SQLi) attacks from safe queries. The implementation utilized a reinforcement learning framework, in which a DQN agent interacted with a simulated environment (SQLiEnv), learned from experience through reward-based feedback, and refined its prediction policy over multiple episodes.

The training process was conducted over 2,000 episodes, with each episode consisting of 100 interactions. For each interaction, the agent was either rewarded (+1) for correct predictions or penalized (–1) for incorrect ones.

The learning process was guided by reinforcement principles such as experience replay and epsilon-greedy exploration, allowing the agent to explore new actions early in training while gradually focusing on exploiting learned policies as training progressed.

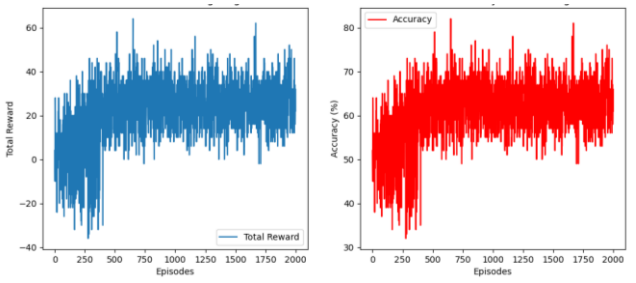

Fig. 4. Reinforcement learning progressions (left) & model accuracy over training (right).

As shown in Fig. 4, the total reward (left graph) displayed a notable upward trend in the early episodes, although with some fluctuations—particularly during the exploration phase when epsilon was still high. After approximately 500 episodes, both reward and accuracy metrics showed significant improvements, stabilizing around *Episodes 600 to 800*. The right-hand side of the figure reveals the *accuracy curve*, which began in the 40–50% range and climbed steadily to reach peak values of up to 82%, with a sustained accuracy range between 63–73% toward the end of the training cycle. Additionally, Table III presents the Training Progressions across 2000 episodes and to better understand this progression, Table IV categorizes the model's development across four key training phases.

TABLE III. TRAINING PHASES OF THE DQN MODEL

Training Phase	Epsilon Range	Accuracy Trend	Notable Highlights
Early Exploration	1.0 → ~0.6	43%–57%	Inconsistent learning; mixed performance
Learning Growth	~0.6 → 0.2	60%–72%	First signs of reliable detection
Policy Exploitation	0.2 → 0.01	63%–82%	Peak accuracy, stable and high performance
Final Convergence	Steady at 0.01	65%–73% (avg. sustained)	Long-term generalization and robustness

Table III further illustrates how the epsilon decay mechanism guided the agent's transition from exploration to exploitation. In the Early Exploration phase, the model exhibited erratic behavior as it attempted to learn the structure of the input data. As the epsilon value decreased, the model entered the Learning Growth stage, where it started making increasingly accurate classifications. The Policy Exploitation phase, characterized by a low epsilon, allowed the agent to rely on learned behavior with minimal randomness. Finally, in the Final Convergence phase, the model achieved sustained, stable performance with an average accuracy consistently above 65%.

Hence, these indicators significantly provided a comprehensive understanding of this research that aimed to propose, develop, integrate and evaluate the model's accuracy, adaptability, and performance across multiple episodes was confirmed using labeled dataset. It was also strengthen the researcher's observation by applying and finding the optimal hyperparameters in analyzing the model's behavior over 2,000 training episodes, with three key performance indicators tracked: Accuracy, Total Reward (Reward), Epsilon Decay (exploration rate/decay) and Performance Stability (model stability across epochs/episodes) as presented in Table IV.

TABLE IV. MODEL EVALUATION

Performance Metrics	Observation/Evaluation Interpretation
Accuracy	43% (early) → 73% (final average), peaked at 82%
Reward	–22 (low point) → +64 (high point), stable at +30–40 range
Epsilon Decay	1.0 → 0.01, indicating improved policy confidence
Performance Stability	Stabilized from Episode ~600 onwards

Hence, these indicators significantly provided a comprehensive understanding of the DQN model's learning effectiveness and generalization ability in classifying SQLi queries.

V. CONCLUSION

This study evidently provided the effectiveness of a DQN-based reinforcement learning model in detecting SQLi attacks within structured web query patterns. The model then exhibited a clear learning trajectory—starting with unstable predictions and evolving toward sustained classification accuracy. Notably, it achieved a peak accuracy of 82% and maintained consistent performance between 65% and 73% across extended episodes, affirming its capacity for pattern recognition and generalization.

The sustained gains in reward and accuracy reflect a successful convergence and an optimized policy that effectively differentiated between malicious and benign SQL statements. The findings notably contributed to the theoretical advancement of intelligent intrusion detection systems by validating reinforcement learning's adaptive capabilities in cybersecurity contexts. Unlike traditional models, which often rely on static features or handcrafted rules, the DQN framework leveraged dynamic policy updates and experience replacing iteratively improve its classification strategy. These results also align with existing literature highlighting the importance of policy-based agents in real-time threat mitigation and expand prior works by demonstrating DQN's capacity for maintaining long-term accuracy over diverse query structures.

Furthermore, this research highlights the relevance of reinforcement learning for evolving cyber threats and affirms the model's applicability in practical deployment scenarios. The model's consistent performance across a diverse dataset suggests its potential for integration into adaptive security layers of web systems, where real-time learning and response are crucial. Overall, this study provides empirical evidence that a DQN-based model, when properly tuned and trained, can serve as a robust, intelligent mechanism for mitigating SQLi attacks, thereby enhancing the theoretical discourse on automated and interpretable cybersecurity solutions.

VI. RECOMMENDATIONS

Based on the findings of this research, it is recommended that future studies focus on enhancing the data preprocessing pipeline to further improve model performance. Although character-level encoding and rule-based SQLi classification supported the detection of malicious queries, the application of more advanced natural language processing (NLP), GPT-4 embeddings, Quantum Machine Learning approaches or in multi-modal frameworks that cover similar techniques, including tokenization, word embeddings, and syntactic parsing, may enable the model to better recognize obfuscated or sophisticated SQL injection attempts. Incorporating sequence modeling methods, such as bidirectional encoders, could also strengthen the contextual understanding of logical query structures, leading to more accurate threat detection.

Considering the achieved classification accuracy, additional refinements to the Deep Q-Network architecture are encouraged to optimize both learning efficiency and generalization. While the present network configuration demonstrated consistent performance improvements across training episodes, experimentation with deeper architectures, attention-based mechanisms, and advanced variants such as Double DQN and Dueling DQN is recommended to further enhance the model's resilience against diverse and adversarial input patterns. Furthermore, collecting and curating primary datasets, rather than relying solely on secondary sources, would provide a richer and more realistic foundation for training models capable of adapting to evolving SQL injection techniques. Introducing real-time feedback mechanisms, wherein the model interacts with live web traffic, could also offer dynamic learning opportunities, equipping the system to respond swiftly to emerging threats.

Aligned with the limitations identified in this study, future initiatives should extend beyond the model's detection capabilities and explore the practical integration of the system within application environments. As this research was confined to model development and evaluation, without designing a user interface or a complete deployment framework, subsequent efforts should address the operationalization of the model to ensure usability and scalability in production settings. Moreover, since this study did not delve into the identification or categorization of specific query structures such as subqueries, inner queries, scalar queries, column queries, row queries, and table queries, future research could investigate techniques for parsing and analyzing internal SQL query compositions to achieve finer-grained threat classification.

Lastly, it is recommended that the developed model be evaluated under live operational conditions to thoroughly assess its robustness, adaptability, and scalability across diverse database systems and application environments. Validation across varying technological contexts is crucial to ensure the model's generalizability and practical effectiveness in real-world scenarios. Future research may also consider expanding the scope to address other types of injection attacks beyond SQL injection, implement on machine learning embeddings visualizations, hence this broadens the model's understandability, applicability and complexity to a wider range of cybersecurity threats. Integrating the model into comprehensive security frameworks would further contribute to strengthening system defenses and enhancing overall resilience against evolving vulnerabilities.

REFERENCES

[1] N. Salih and A. Samad, "Protection Web Applications using Real-Time Technique to Detect Structured Query Language Injection Attacks," Int. J. Comput. Appl., vol. 149, no. 6, pp. 26–32, 2016, doi: 10.5120/ijca2016911424.

[2] H. Furhad, R. K. Chakrabortty, M. J. Ryan, J. Uddin, and I. H. Sarker, "A hybrid framework for detecting structured query language injection attacks in web-based applications," Int. J. Electr. Comput. Eng., vol. 12, no. 5, pp. 5405–5414, 2022, doi: 10.11591/ijece.v12i5.pp5405-5414.

[3] N. S. Ali, "Investigation framework of web applications vulnerabilities, attacks and protection techniques in structured query language injection attacks," Int. J. Wirel. Mob. Comput., vol. 14, no. 2, pp. 103–122, 2018, doi: 10.1504/IJWMC.2018.091137.

[4] Z. Lu, "Sql injection detection using Naïve Bayes classifier : a probabilistic approach for web application security," vol. 04016, 2025.

[5] W. B. Demilie and F. G. Deriba, "Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques," J. Big Data, vol. 9, no. 1, 2022, doi: 10.1186/s40537-022-00678-0.

[6] A. Odeh and A. A. Taleb, "Ensemble learning techniques against structured query language injection attacks," Indones. J. Electr. Eng. Comput. Sci., vol. 35, no. 2, pp. 1004–1012, 2024, doi: 10.11591/ijeecs.v35.i2.pp1004-1012.

[7] E. Peralta-Garcia, J. Quevedo-Monsalbe, V. Tuesta-Monteza, and J. Arcila-Diaz, "Detecting Structured Query Language Injections in Web Microservices Using Machine Learning," Informatics, vol. 11, no. 2, 2024, doi: 10.3390/informatics11020015.

[8] Y. Guan, J. He, T. Li, H. Zhao, and B. Ma, "SSQLi: A Black-Box Adversarial Attack Method for SQL Injection Based on Reinforcement Learning," Futur. Internet, vol. 15, no. 4, 2023, doi: 10.3390/fi15040133.

[9] M. Hasan, A. Al-Maliki, and N. Jasim, "Review of SQL injection attacks: Detection, to enhance the security of the website from client-side attacks," Int. J. Nonlinear Anal. Appl, vol. 13, no. October 2021, pp. 2008–6822, 2022, [Online]. Available: http://dx.doi.org/10.22075/ijnaa.2022.6152

[10] A. M. Ahmed, "The Scientific Journal of Cihan University – Sulaimaniya," Sci. J. Cihan Univ. – Sulaimaniya, vol. 6, no. 1, pp. 145–156, 2022.

[11] M. Abdulridha Hussain et al., "Provably throttling SQLI using an enciphering query and secure matching," Egypt. Informatics J., vol. 23, no. 4, pp. 145–162, 2022, doi: https://doi.org/10.1016/j.eij.2022.10.001.

[12] S. M. Shagari, D. Gabi, N. M. Dankolo, and N. N. Gana, "Countermeasure to Structured Query Language Injection Attack for Web Applications using Hybrid Logistic Regression Technique," J. Niger. Soc. Phys. Sci., vol. 4, no. 4, pp. 1–8, 2022, doi: 10.46481/jnsps.2022.832.

[13] V. Abdullayev and A. S. Chauhan, "SQL Injection Attack: Quick View," Mesopotamian J. CyberSecurity, vol. 2023, pp. 30–34, 2023, doi: 10.58496/MJCS/2023/006.

[14] J. Ramírez, W. Yu, and A. Perrusquía, Model-free reinforcement learning from expert demonstrations: a survey, vol. 55, no. 4. 2022. doi: 10.1007/s10462-021-10085-1.

[15] E. Ginzburg-Ganz et al., "Reinforcement Learning Model-Based and Model-Free Paradigms for Optimal Control Problems in Power Systems: Comprehensive Review and Future Directions," Energies, vol. 17, no. 21, 2024, doi: 10.3390/en17215307.

[16] H. Kheddar, D. W. Dawoud, A. I. Awad, Y. Himeur, and M. K. Khan, "Reinforcement-Learning-Based Intrusion Detection in Communication Networks: A Review," IEEE Commun. Surv. Tutorials, p. 1, 2024, doi: 10.1109/COMST.2024.3484491.

[17] Zabeehullah et al., "DQQS: Deep Reinforcement Learning-Based Technique for Enhancing Security and Performance in SDN-IoT Environments," IEEE Access, vol. 12, pp. 60568–60587, 2024, doi: 10.1109/ACCESS.2024.3392279.

[18] H. Alavizadeh, H. Alavizadeh, and J. Jang-Jaccard, "Deep Q-Learning Based Reinforcement Learning Approach for Network Intrusion Detection," Computers, vol. 11, no. 3, pp. 1–19, 2022, doi: 10.3390/computers11030041.

[19] L. Hu, C. Han, X. Wang, H. Zhu, and J. Ouyang, "Security Enhancement for Deep Reinforcement Learning-Based Strategy in Energy-Efficient Wireless Sensor Networks," Sensors, vol. 24, no. 6, pp. 1–14, 2024, doi: 10.3390/s24061993.

[20] U. Habib, "A Survey on Implication of Artificial Intelligence in detecting SQL Injections International Journal of Computer and Applications A Survey on Implication of Artificial Intelligence in detecting SQL Injections," Artic. Int. J. Comput. Appl., no. February, 2024, [Online]. Available: https://www.researchgate.net/publication/378496266

[21] S. T. Hossain, T. Yigitcanlar, K. Nguyen, and Y. Xu, "Local Government Cybersecurity Landscape: A Systematic Review and Conceptual Framework," Appl. Sci., vol. 14, no. 13, 2024, doi: 10.3390/app14135501.

[22] S. Bamohabbat Chafjiri, P. Legg, J. Hong, and M.-A. Tsompanas, "Vulnerability detection through machine learning-based fuzzing: A systematic review," Comput. Secur., vol. 143, p. 103903, 2024, doi: https://doi.org/10.1016/j.cose.2024.103903.

[23] J. R. Tadhani, V. Vekariya, V. Sorathiya, S. Alshathri, and W. El-Shafai, "Securing web applications against XSS and SQLi attacks using a novel deep learning approach," Sci. Rep., vol. 14, no. 1, pp. 1–17, 2024, doi: 10.1038/s41598-023-48845-4.

[24] R. R. Choudhary, S. Verma, and G. Meena, "Detection of SQL Injection attack Using Machine Learning," 2021 IEEE Int. Conf. Technol. Res. Innov. Betterment Soc. TRIBES 2021, 2021, doi: 10.1109/TRIBES52498.2021.9751616.

[25] H. Sun, Y. Du, and Q. Li, "Deep Learning-Based Detection Technology for SQL Injection Research and Implementation," Appl. Sci., vol. 13, no. 16, 2023, doi: 10.3390/app13169466.

[26] S. Islam, "Future Trends in Sql Databases and Big Data Analytics: Impact of Machine Learning and Artificial Intelligence," Int. J. Sci. Eng., vol. 1, no. 4, pp. 47–62, 2024, doi: 10.62304/ijse.v1i04.188.

[27] A. Khan, K. Khan, W. Khan, S. N. Khan, and R. Haq, "Knowledge-based Word Tokenization System for Urdu," J. Informatics Web Eng., vol. 3, no. 2, pp. 86–97, 2024, doi: 10.33093/jiwe.2024.3.2.6.

[28] R. L. Alaoui and E. H. Nfaoui, "Web attacks detection using stacked generalization ensemble for LSTMs and word embedding," Procedia Comput. Sci., vol. 215, pp. 687–696, 2022, doi: https://doi.org/10.1016/j.procs.2022.12.070.

[29] F. Jáñez-Martino, R. Alaiz-Rodríguez, V. González-Castro, E. Fidalgo, and E. Alegre, "Classifying spam emails using agglomerative hierarchical clustering and a topic-based approach," Appl. Soft Comput., vol. 139, p. 110226, 2023, doi: https://doi.org/10.1016/j.asoc.2023.110226.

[30] E. Sk, Text Representation Methods for Big Social Data.

[31] W. Hsieh et al., "Deep Learning, Machine Learning -- Digital Signal and Image Processing: From Theory to Application," 2024, [Online]. Available: http://arxiv.org/abs/2410.20304

[32] K. C. Santos, R. S. Miani, and F. de Oliveira Silva, "Evaluating the Impact of Data Preprocessing Techniques on the Performance of Intrusion Detection Systems," J. Netw. Syst. Manag., vol. 32, no. 2, p. 36, 2024, doi: 10.1007/s10922-024-09813-z.

[33] A. Zahid, "VULNERABILITY DETECTION AND PREVENTION : AN APPROACH TO ENHANCE CYBERSECURITY," no. August, 2024, doi: 10.13140/RG.2.2.31687.71841.

[34] I. A. Shah, N. Z. Jhanjhi, and S. N. Brohi, "Proposing Model for Classification of Malicious SQLi Code Using Machine Learning Approach," 1st Int. Conf. Innov. Eng. Sci. Technol. Res. ICIESTR 2024 - Proc., pp. 1–5, 2024, doi: 10.1109/ICIESTR60916.2024.10798230.

[35] A. Kumar, P. Nagarkar, P. Nalhe, and S. Vijayakumar, "Deep Learning Driven Natural Languages Text to SQL Query Conversion: A Survey," vol. 14, no. 8, pp. 1–18, 2022, [Online]. Available: http://arxiv.org/abs/2208.04415

[36] T. Houichime and Y. El Amrani, "Context Is All You Need: A Hybrid Attention-Based Method for Detecting Code Design Patterns," IEEE Access, vol. 13, pp. 9689–9707, 2025, doi: 10.1109/ACCESS.2025.3525849.

[37] H. Salem, H. Salloum, O. Orabi, K. Sabbagh, and M. Mazzara, "Enhancing News Articles: Automatic SEO Linked Data Injection for Semantic Web Integration," Appl. Sci., vol. 15, no. 3, pp. 1–18, 2025, doi: 10.3390/app15031262.

[38] C. Zhao, X. Yuan, J. Long, L. Jin, and B. Guan, "Chinese stock market Pr ep rin t n ot pe er re v Pr ep rin t n ot pe er re v ed".

[39] V. Franzoni, S. Tagliente, and A. Milani, "Generative Models for Source Code: Fine-Tuning Techniques for Structured Pattern Learning," Technologies, vol. 12, no. 11, pp. 1–21, 2024, doi: 10.3390/technologies12110219.

[40] M. Sewak, S. K. Sahay, and H. Rathore, "Deep Reinforcement Learning in the Advanced Cybersecurity Threat Detection and Protection," Inf. Syst. Front., vol. 25, no. 2, pp. 589–611, 2023, doi: 10.1007/s10796-022-10333-x.

[41] Z. Qiu, Y. Tao, S. Pan, and A. W. C. Liew, "Knowledge Graphs and Pretrained Language Models Enhanced Representation Learning for Conversational Recommender Systems," IEEE Trans. Neural Networks Learn. Syst., vol. 14, no. 8, pp. 1–15, 2024, doi: 10.1109/TNNLS.2024.3395334.

[42] M. S. Hamidi and M. Doostari, "Automated Multi-Step Web Application Attack Analysis Using Reinforcement Learning and Vulnerability Assessment Tools," 2023.

[43] A. A. Hammad, S. R. Ahmed, M. K. Abdul-Hussein, M. R. Ahmed, D. A. Majeed, and S. Algburi, "Deep Reinforcement Learning for Adaptive Cyber Defense in Network Security," in Proceedings of the Cognitive Models and Artificial Intelligence Conference, in AICCONF '24. New York, NY, USA: Association for Computing Machinery, 2024, pp. 292–297. doi: 10.1145/3660853.3660930.

[44] E. Prediction, "Spectrum of Engineering Sciences," vol. 3, no. 2, pp. 272–303, 2025.

[45] M. S. Ramzan et al., "Spectrum of Engineering Sciences," vol. 3, no. 2, pp. 90–125, 2025.

[46] K. Salah Fathi, S. Barakat, and A. Rezk, "An effective SQL injection detection model using LSTM for imbalanced datasets," Comput. Secur., vol. 153, p. 104391, 2025, doi: https://doi.org/10.1016/j.cose.2025.104391.

[47] A. Sharma, V. G. K. Kumar, and A. Poojari, "Prioritize Threat Alerts Based on False Positives Qualifiers Provided by Multiple AI Models

Using Evolutionary Computation and Reinforcement Learning," J. Inst. Eng. Ser. B, 2024, doi: 10.1007/s40031-024-01175-z.

[48] M. Vasconcelos and L. Cavique, "Mitigating false negatives in imbalanced datasets: An ensemble approach," Expert Syst. Appl., vol. 262, p. 125674, 2025, doi: https://doi.org/10.1016/j.eswa.2024.125674.

[49] J. H. Cabot and E. G. Ross, "Evaluating prediction model performance," Surgery, vol. 174, no. 3, pp. 723–726, 2023, doi: https://doi.org/10.1016/j.surg.2023.05.023.

[50] G. Naidu, T. Zuva, and E. M. Sibanda, "A Review of Evaluation Metrics in Machine Learning Algorithms," in Artificial Intelligence Application in Networks and Systems, R. Silhavy and P. Silhavy, Eds., Cham: Springer International Publishing, 2023, pp. 15–25.

[51] S. Niu, X. Pan, J. Wang, and G. Li, "Deep reinforcement learning from human preferences for ROV path tracking," Ocean Eng., vol. 317, p. 120036, 2025, doi: https://doi.org/10.1016/j.oceaneng.2024.120036.

[52] A. Corrêa, A. Jesus, C. Silva, P. Peças, and S. Moniz, "Rainbow Versus Deep Q-Network: A Reinforcement Learning Comparison on The Flexible Job-Shop Problem," IFAC-PapersOnLine, vol. 58, no. 19, pp. 870–875, 2024, doi: https://doi.org/10.1016/j.ifacol.2024.09.176.

[53] Z. Dai and Y. Zhang, "DJAYA-RL: Discrete JAYA algorithm integrating reinforcement learning for the discounted {0-1} knapsack problem," Swarm Evol. Comput., vol. 95, p. 101927, 2025, doi: https://doi.org/10.1016/j.swevo.2025.101927.

[54] N. Trabelsi, L. Chaari Fourati, and W. Jaafar, "Deep reinforcement learning for autonomous SideLink radio resource management in platoon-based C-V2X networks: An overview," Comput. Networks, vol. 255, p. 110901, 2024, doi: https://doi.org/10.1016/j.comnet.2024.110901.

[55] J. B. Rola et al., "Convolutional Neural Network Model for Cacao Phytophthora Palmivora Disease Recognition," Int. J. Adv. Comput. Sci. Appl., vol. 15, no. 8, pp. 986–990, 2024, doi: 10.14569/IJACSA.2024.0150897.

[56] H. Babbar, S. Rani, and M. Driss, Effective DDoS attack detection in software-defined vehicular networks using statistical flow analysis and machine learning, vol. 19, no. 12. 2024. doi: 10.1371/journal.pone.0314695.

[57] S. Zhou, C. Liu, D. Ye, T. Zhu, W. Zhou, and P. S. Yu, "Adversarial Attacks and Defenses in Deep Learning: From a Perspective of Cybersecurity," ACM Comput. Surv., vol. 55, no. 8, Dec. 2022, doi: 10.1145/3547330.

[58] V. Kumar, N. Kedam, K. V. Sharma, D. J. Mehta, and T. Caloiero, "Advanced Machine Learning Techniques to Improve Hydrological Prediction: A Comparative Analysis of Streamflow Prediction Models," Water (Switzerland), vol. 15, no. 14, 2023, doi: 10.3390/w15142572.

[59] "SQL Injection Dataset." https://www.kaggle.com/datasets/sajid576/sql-injection-dataset?resource=download

[60] J. M. Gorriz, F. Segovia, J. Ramirez, A. Ortiz, and J. Suckling, "Is K-fold cross validation the best model selection method for Machine Learning?," 2024, [Online]. Available: http://arxiv.org/abs/2401.16407

[61] A. Seraj et al., "Chapter 5 - Cross-validation," in Handbook of Hydroinformatics, S. Eslamian and F. Eslamian, Eds., Elsevier, 2023, pp. 89–105. doi: https://doi.org/10.1016/B978-0-12-821285-1.00021-X.

[62] W. Shen, W. Lin, W. Wu, H. Wu, and K. Li, "Reinforcement learning-based task scheduling for heterogeneous computing in end-edge-cloud environment," Cluster Comput., vol. 28, no. 3, 2025, doi: 10.1007/s10586-024-04828-2.

[63] J. Liu et al., "HeterPS: Distributed deep learning with reinforcement learning based scheduling in heterogeneous environments," Futur. Gener. Comput. Syst., vol. 148, pp. 106–117, 2023, doi: 10.1016/j.future.2023.05.032.

[64] K. Galbraith, O. Alaca, A. R. Ekti, A. Wilson, I. Snyder, and N. M. Stenvig, "On the Investigation of Phase Fault Classification in Power Grid Signals: A Case Study for Support Vector Machines, Decision Tree and Random Forest," 2023 North Am. Power Symp. NAPS 2023, no. Ll, 2023, doi: 10.1109/NAPS58826.2023.10318740.

[65] J. Program and S. Pendidikan, "3 1,2*,3," vol. 12, no. 1, pp. 1309–1321, 2023.

[66] M. Bansal, A. Goyal, and A. Choudhary, "A comparative analysis of K-Nearest Neighbor, Genetic, Support Vector Machine, Decision Tree, and Long Short Term Memory algorithms in machine learning," Decis. Anal. J., vol. 3, p. 100071, 2022, doi: https://doi.org/10.1016/j.dajour.2022.100071.

[67] B. Mondal, A. Banerjee, and S. Gupta, "review of SQLI detection strategies using machine learning," Int. J. Health Sci. (Qassim)., vol. 6, no. May, pp. 9663–9676, 2022, doi: 10.53730/ijhs.v6ns2.7519.

[68] G. Ali and M. M. Mijwil, "Cybersecurity for Sustainable Smart Healthcare: State of the Art, Taxonomy, Mechanisms, and Essential Roles," Mesopotamian J. CyberSecurity, vol. 4, no. 2, pp. 20–62, 2024, doi: 10.58496/mjcs/2024/006.

[69] N. M. Aris, N. H. Ibrahim, and N. D. A. Halim, "Design and Development Research (DDR) Approach in Designing Design Thinking Chemistry Module to Empower Students' Innovation Competencies," J. Adv. Res. Appl. Sci. Eng. Technol., vol. 44, no. 1, pp. 55–68, 2025, doi: 10.37934/araset.44.1.5568.

[70] C. J. P. Abuda and R. S. Villafuerte, "Development of an Algorithm-Based Analysis-Compression Integrated Communication Tracking Management Information System (iCTMIS)," Int. J. Adv. Comput. Sci. Appl., vol. 16, no. 3, pp. 107–118, 2025, doi: 10.14569/ijacsa.2025.0160311.

[71] X. Zhang, S. Lv, M. Xu, and W. Mu, "Applying evolutionary prototyping model for eliciting system requirement of meat traceability at agribusiness level," Food Control, vol. 21, no. 11, pp. 1556–1562, 2010, doi: https://doi.org/10.1016/j.foodcont.2010.03.020.

[72] R. A. Carter, A. I. Anton, A. Dagnino, and L. Williams, "Evolving beyond requirements creep: a risk-based evolutionary prototyping model," in Proceedings Fifth IEEE International Symposium on Requirements Engineering, 2001, pp. 94–101. doi: 10.1109/ISRE.2001.948548.

[73] C. J. P. Abuda and R. S. Villafuerte, "Development of an Algorithm-Based Analysis-Compression Integrated Communication Tracking Management Information System (iCTMIS)," 2024 IEEE Open Conf. Electr. Electron. Inf. Sci. eStream 2024 - Proc., 2024, doi: 10.1109/eStream61684.2024.10542580.

[74] L. L. Pullum, "Review of Metrics to Measure the Stability, Robustness and Resilience of Reinforcement Learning," pp. 59–78, 2023, doi: 10.5121/csit.2023.130205.