Machine Learning in SQL Injections Detection

Armando Valdez
valde148@morris.umn.edu
Division of Science and Mathematics
University of Minnesota, Morris
Morris, Minnesota, USA

Abstract

As SQL injection (SQLi) techniques grow more complex, traditional rule-based detection methods are increasingly ineffective. Attackers now use advanced tools to bypass static filters and signature-based Web Application Firewalls (WAFs), exposing major gaps in web security. This paper explores how machine learning (ML) models have become essential for detecting unpredictable SQLi threats. Analyzing Improved Text-CNN and XploitSQL, it highlights ML's role in both defending against and generating advanced attacks. Detection models like Improved Text-CNN recognize subtle patterns in SQL queries, while adversarial tools like XploitSQL show how machine learning can also be used to evade defenses. Together, they reflect a growing machine learning arms race in fighting SQL injections.

Keywords: SQLi, Machine Learning, Deep Learning, Queries

1 Introduction

SQL injection (SQLi) is a hacking technique where malicious SQL statements are inserted into web input fields to manipulate database queries. Despite increased awareness, SQLi remains one of the most dangerous threats according to the Open Web Application Security Project (OWASP). Traditional defenses—such as input validation, query filters, and rule-based Web Application Firewalls (WAFs)—often fail to adapt to modern obfuscation techniques. As a response, machine learning (ML) has emerged as a powerful tool capable of learning complex attack patterns and improving SQLi detection [5, 12]. This paper explores the role of machine learning in both detecting and generating SQLi attacks. It begins by providing background on how SQLi works and why traditional defenses fall short. Then, it explains how machine learning models are trained, focusing on neural networks and reinforcement learning. The structure of the Improved Text-CNN model is detailed, highlighting its components like embeddings, convolutional layers, and attention mechanisms. Next, the paper analyzes the adversarial architecture of XploitSQL using workflow diagrams to show how it generates evasive payloads. Both models are evaluated through testing frameworks that assess their accuracy and evasion capabilities. The paper then compares the strengths and weaknesses of each approach before concluding with reflections on the future of adaptive ML-based cybersecurity solutions.

2 Background

2.1 SQL Injection Techniques

SQL stands for Structured Query Language, and is a programming language that is used to command databases. Queries are used to make simple commands used to reorganize, take or insert data from databases [12]. SQLi are attacks on SQL queries that I discussed in the Introduction. A common target for these attacks is a login form, where an attacker can manipulate input fields to trick the application into logging them in without valid credentials. This is a type of *in-band* SQL injection, where the attacker uses the same communication channel to both send the malicious input and receive the results, unlike *out-of-band* SQL injection, which is the opposite of in-band, that sends data back through a different path than the one used to inject the payload [6]. One common form of in-band SQLi is the *tautology-based* attack, which works by injecting conditions that are always true. For example, injecting a statement like OR 1 = 1 into a password field changes the logic of the query so that the attacker can successfully log in without knowing the password. This allows the attacker to bypass authentication checks entirely [4].

2.2 Example of a Simple SQL Injection Attack

Suppose in a login system the backend query looks for a user with a matching username and password. If an attacker enters admin as the username and any symbols followed by OR 1 = 1 as the password, the query logic is changed. Instead of checking whether the password is correct, it evaluates whether 1 = 1, which is always true. As a result, the application grants access without validating the actual password [4, 6].

2.3 Limitations of Traditional SQL Injection Detection Techniques

Signature-based WAFs compare inputs to a list of known attack patterns, while rule-based systems look for specific SQL keywords or suspicious structures. However, tautology-based injections are easily disguised using variations like OR NOT FALSE, OR A=A, or by inserting comments and special characters. These variations may not match known signatures but still achieve the same malicious effect. *Static detection systems* lack an understanding of semantic equivalence in SQL logic. For example, a rule that flags OR 1=1

may completely miss a more complex variant of OR TRUE, even though both create the same logical bypass [6].

Both the Improved Text-CNN and XploitSQL studies demonstrate that static systems, including commercial WAFs, can be bypassed by intelligently crafted tautology-based injections. XploitSQL, in particular, uses *large language models* (LLMs) combined with *reinforcement learning* to generate highly evasive in-band payloads. These payloads are designed to execute the same tautology logic but in forms that static filters often fail to recognize. Because of these limitations, modern defense strategies are shifting toward machine learning–based models, which can learn and generalize from large datasets of known and mutated attacks.

2.4 Machine Learning and Subsets of ML

Machine learning (ML) is a system that allows computers to learn patterns from data and make decisions or predictions without being explicitly programmed. Instead of using fixed rules, ML models adapt by analyzing large datasets, identifying trends, and improving performance over time [8].

Neural networks (NN), a subset of ML, are computer systems made of layers of connected nodes (called neurons) that process data step by step. Each layer transforms the input into patterns and passes it to the next. The input layer takes in raw data, hidden layers extract important features, and the output layer gives the final result. Weights that connect inputs, layers, and outputs control how important each input is when making a prediction, and they get adjusted during training to help the model improve. A neural network takes the incoming values, multiplies them by weights, adds them up to another parameter called bias, and passes the result through a mathematical function known as an activation function. Neural networks are trained by showing them many labeled (i.e containing the correct answer that the model is trying to learn) examples and using a method called backpropagation to adjust their internal weights. This means they learn over time which patterns lead to which outcomes, getting better at making correct predictions [8].

For binary classification tasks, such as identifying whether a SQL query is normal or malicious, the output layer typically contains two neurons, and their outputs are passed through a Softmax function to convert them into class probabilities. The softmax function is a formula that measures probabilities from 0 to 1 [12].

Deep learning (DL), a subset of machine learning, uses multi-layered neural networks to automatically learn and extract complex patterns from data. Unlike earlier models that relied on manually selected features, deep learning models can process raw input and gradually learn meaningful representations through layers [6].

Reinforcement Learning (RL) is a subset of ML and is used in the XploitSQL system to train a language model to generate smarter SQL injection attacks. The model learns by trial and error, directly interacting with the environment it operates in, in this case web systems [3]. It also creates SQL queries, and if those queries are valid, malicious, and successfully bypass detection systems, it receives a higher reward. Over time, the model uses this feedback to improve, making its attacks more evasive and harder for security tools to catch.

2.5 Subsets of Deep Learning

This section discusses subsets of deep learning, including *Convolutional Neural Networks* (CNNs) and *Large Language Models* (LLMs), both have shown remarkable success in *natural language processing* (NLP).

CNNs are deep learning models that use *convolutional filters* to scan input text and automatically detect local patterns, such as short phrases or logic operators commonly found in SQL injection attacks [12]. They are especially effective in tasks like text classification, where the model must determine whether a given query is benign or malicious.

Large Language Models (LLMs) are another subset of DL model, trained on massive datasets to understand and generate human-like text. In the XploitSQL framework, an LLM is fine-tuned to generate SQL injection queries by learning the structure and logic of actual SQL statements. It serves as the base generator, producing a variety of SQLi payloads, which are then refined through reinforcement learning to become more effective at bypassing detection systems [9].

3 Training the ML Models

3.1 Training Neural Networks

Neural networks (NN) and deep learning (DL) models are made of layers of connected units that process input, extract features, and make predictions. In SQL injection detection, input queries are first broken down into tokens—individual words, symbols, or operators. These tokens are converted into numerical representations using word embeddings like *Word2Vec*, which capture semantic relationships by placing tokens used in similar ways closer together in vector space. For example, logical operators like OR and AND often appear in similar contexts within SQL queries, so their embeddings are positioned near each other. These embeddings are passed through the network, where the model adjusts weights through training on labeled data using backpropagation, gradually learning to detect patterns associated with SQL injection [8, 12].

In the case of the Improved Text-CNN model, convolutional layers are used to detect short, meaningful patterns in the input, while the *Channel Attention Mechanism* (CAM) assigns importance to the most relevant features. Pooling is a process in neural networks that reduces the size of the data while keeping the most important information. After *pooling*, the final set of features is passed to a fully connected layer that produces the final output—a prediction indicating whether the input is a normal query or a SQL injection [12].

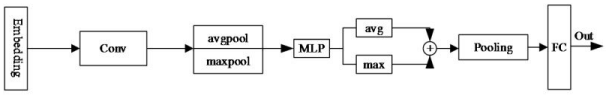

Figure 1. Improved Text-CNN Algorithm Diagram [12].

Backpropagation, which is the process of adjusting weights and biases to minimize the number of incorrect answers on training data, repeats over many iterations, as the network continues to learn from thousands of examples. With each round of training, the network gradually improves, lowering its error and increasing its ability to generalize.

3.2 Training Reinforcement Learning

Reinforcement learning (RL) is a training method where a model learns by interacting with an environment and receiving feedback based on its performance. The model generates outputs—such as examples, queries, or actions—and receives a reward depending on how well the output meets certain goals. Larger rewards encourage behaviors that succeed, while low or no rewards discourage ineffective behaviors. Over time, the model adjusts its internal parameters to favor generating more successful outputs. To keep learning stable, algorithms like Proximal Policy Optimization (PPO) are used, which control how much the model can change at each step, preventing sudden, destabilizing updates. This approach enables the model to gradually improve its performance through trial and error.[5]

4 Structure of Improved Text CNN

4.1 Embedding Layer

In this section, I will go over Figure 1 and explaining the parts of the diagram of the Improved Text-CNN which include Embedding, Conv, avgpool, maxpool, MLP, Pooling, and FC.

The *embedding layer* is the first critical stage in the Improved Text-CNN model, where raw SQL queries are transformed into a format that the NN can process. When a query such as SELECT * FROM users WHERE id = '1' OR '1'='1' enters the model, it begins as plain text [6]. However, neural networks cannot directly interpret words or characters—they require numerical input. To achieve this, the query is first tokenized, meaning it is broken down into individual units such as 'SELECT', '*', 'FROM', 'users', 'OR', and so on. These tokens represent the smallest elements of meaning in the query.

Once tokenized, each word or symbol is converted into a dense vector using the Word2Vec embedding technique. Word2Vec maps each token to a vector where similar words are located near each other in vector space [12].

Without the embedding layer, the model would have no way to understand the meaning or relationships between tokens in a query. By converting words into vectors that carry contextual meaning, the embedding layer lays the foundation for the neural network to recognize signs of SQL injection in both obvious and disguised forms.[12]

4.2 Convolutional Layer (Conv)

This layer is responsible for detecting local patterns in the input, specifically sequences of tokens that may indicate suspicious or malicious behavior. To do this, the model uses convolutional filters, also known as kernels, which are small sliding windows that move across the input text. Each filter looks at a fixed number of consecutive tokens at a time to capture patterns of varying lengths.[12]

As each filter slides across the input text, it performs a *convolution* that computes a weighted combination of the values in the window. This produces a feature map, a new sequence of values that highlights where in the query the filter detected meaningful patterns. For instance, a filter with size 3 might learn to recognize common SQL injection phrases such as "OR 1=1". Importantly, these filters are trained automatically. Each filter becomes a type of "pattern detector" that activates when it sees a specific feature in the query [12].

These feature maps serve as the next input for the *Channel Attention Mechanism (CAM)*, which analyzes and weighs the importance of each pattern before the model proceeds to *pooling* and classification. By the end of the convolutional stage, the model has converted the raw word embeddings into structured features that highlight key token sequences that might indicate an injection attempt.[12]

4.3 Pooling

CAM kicks in to decide which of these maps should have more or less influence on the final prediction. To do that, it uses two specialized pooling operations which are *global max pooling* and *global average pooling*. It also has a small fully connected neural network, also known as a *multi-layer perceptron* (MLP).

Global max pooling takes a single feature map and extracts only its highest value which is the strongest signal that the convolutional filter produced across the entire SQL query. For example, if a filter activates strongly when it sees "OR 1=1".[12]

Global average pooling computes the average of all the values in the same feature map. This gives the model a more balanced view, telling it how consistently important that feature was across the whole query.[12]

Both of these pooled values—one from max pooling and one from average pooling—are then stacked together to create a two-element vector for each feature map. This vector is passed into a small MLP, a basic neural network with one or two layers, which processes this information and outputs a single attention score (a value between 0 and 1). This score is the model's judgment of how important that specific feature map is to the overall task.[12]

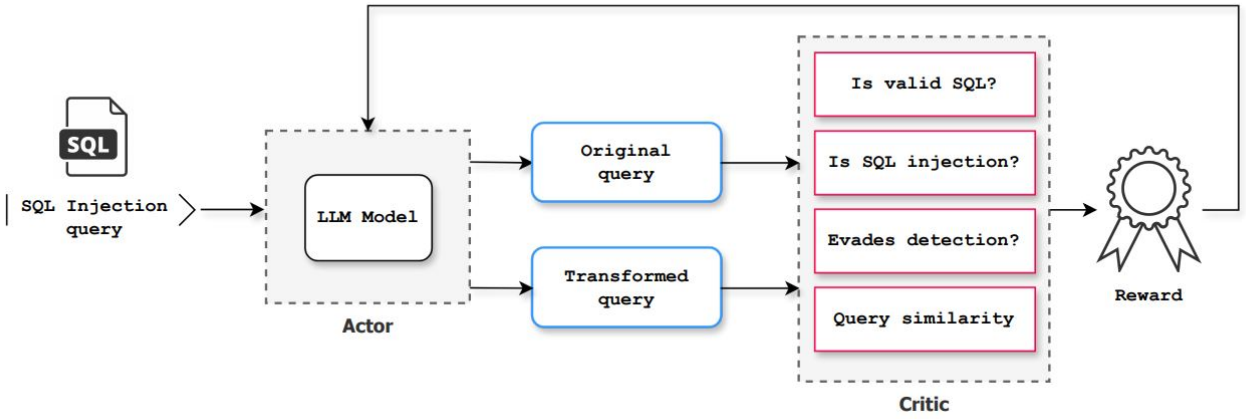

Figure 2. Workflow of LLM and RL models in training XploitSQL [5]

Once the attention score is generated, it is multiplied back onto the original feature map, effectively scaling it. A score near 1 keeps the feature map strong, while a score near 0 dampens it. This process repeats for every feature map produced by the convolutional layer. The end result is a refined set of features where the most relevant patterns are given greater weight in the final decision-making. [12]

4.4 Fully Connected Layer

Once the feature maps have been refined by CAM the model compresses each one down to a single value using global max pooling, producing a flat feature vector. This vector represents the most important patterns detected across all filters, a summary of what the model has learned from the SQL query. This summary is then passed into the *fully connected* (FC) layer, which means a layer where every input feature is connected to every output neuron. The FC layer combines all the extracted and weighted features using learned weights and biases to compute raw prediction scores, also called logits. These scores represent how strongly the model believes the input belongs to each possible class. In this case, there are two classes: a normal SQL query or a SQL injection.[12]

However, these raw scores are not yet interpretable as probabilities. That's where the Softmax function comes in. Softmax takes the logits produced by the FC layer and transforms them into a probability distribution across the output classes. It scales the values so that they sum to 1, making them easier to interpret. For example, if the model outputs logits like [1.2, 3.8], the Softmax layer might convert them into probabilities like [0.08, 0.92], meaning the model is 92 percent confident the query is a SQL injection. This probability becomes the model's final prediction. If the SQLi class has the highest score, the model classifies the query as malicious; otherwise, it is marked as normal.[12]

5 Structure of XploitSQL

5.1 Large Language Model and Actor

In this section, I will go over Figure 2 and explain what Actor, Critic, Reward, and how they relate to the workflow of XploitSQL.

XploitSQL is a system designed to generate SQLi payloads by combining LLMs with reinforcement learning techniques. LLMs are deep learning models trained on massive amounts of text data to understand and generate human-like language [9]. In this system, an LLM is used to create initial SQL queries, and reinforcement learning is applied to refine them into more evasive and effective attacks. In the case of XploitSQL, the LLM is fine-tuned specifically on SQL syntax and injection patterns, allowing it to generate syntactically valid and potentially malicious SQL queries.

In reinforcement learning terminology, the *Actor* is the component that makes decisions or takes actions. In this system, the Actor is the LLM itself, and its action is to generate a SQLi payload (an attempted SQL injection query) based on a given prompt or context. This could be a natural language instruction like "generate a login bypass," or it could be an abstract representation of the SQL injection task [5].

When the LLM receives this input, it uses what it has learned during pretraining and fine-tuning to produce the entire query, token by token. This process is probabilistic: the model selects words based on learned patterns and probabilities, forming queries such as OR 1=1. At this early stage, the LLM isn't yet optimized to evade detection, it simply produces what it "knows" from training.

That's where reinforcement learning comes in next to refine the LLM so that it learns to generate payloads that are not only valid but also capable of bypassing detection systems [5, 9].

5.2 Critic

The goal is to determine whether the LLM successfully generated a query that is malicious, executable, and capable of evading detection. Each generated query is referred to as a transformed query, meaning it has evolved from the LLM's learned patterns into a potentially harmful SQL injection attempt. The evaluation process is overseen by the *Critic*, a key component in reinforcement learning that assigns a numerical score (or reward) based on how effective or ineffective the LLM's generated query was [3].

The first part of the evaluation asks, "Is it valid SQL?" This step ensures that the generated payload is syntactically correct and can be executed by a database without producing an error. Even if a query contains injection logic, it won't be useful if it causes a syntax error.

Next, the system checks, "Is it a SQL injection?" This means asking whether the transformed query changes the logic of the original query in a malicious way. A query that introduces such logic is considered a successful SQL injection. This step ensures that the model isn't just producing harmless or decorative changes, but is actively learning to break the intended behavior of the query.

The third and most important test is, "Does it evade detection?" This is where the model is evaluated against existing SQL injection defenses, including both WAF and machine learning models detectors like CNNs. If the transformed query successfully bypasses these defenses, it receives a high reward. If it is flagged or blocked, the reward is lowered [5].

Finally, the system performs a query similarity check. This step measures how different the transformed query is from the original input. A high similarity score would mean the LLM generated a query that is too close to what it already knows, suggesting it is not creating sufficiently new or creative payloads. In contrast, a more substantially modified query that still works evades detection, and receives a higher reward is preferred. This discourages the model from making only superficial changes like altering spaces or punctuation. Instead, it pushes the model to invent new variations of SQL injection that are both functional and harder to detect.

5.3 Reward

A high reward means the payload was valid, executed an injection, and bypassed security systems. A low reward signals failure because it might be that the query was invalid, not malicious, or got caught by a detector. This reward becomes the key feedback that guides how the LLM improves its future generations.

To process this feedback, XploitSQL uses PPO to help the model learn what works without overcorrecting based on just one example. When the model receives a reward for a specific output, PPO adjusts the probabilities the model assigns to similar actions, in this case, generating similar query structures [5].

In the XploitSQL system, the loop of generation, evaluation, reward, and PPO-based learning enables the model to behave like an adaptive, evolving attacker. But beyond technical performance, the real value lies in what it reveals to defenders [9].

6 Testing Models

6.1 Results for Improved Text-CNN

To evaluate the performance of the Improved Text-CNN model, experiments were conducted using a dataset generated from *LibInjection*, a widely used SQL injection detection library available on GitHub. This dataset included a balanced mix of malicious SQL injection queries and normal SQL statements, allowing the model to learn clear distinctions between safe and harmful inputs. The data was divided into three parts: a *training set* used to teach the model, a *validation set* to tune model parameters, and a *test set* used to evaluate the model's final performance on new, unseen in the training and validation SQL code [5].

The evaluation focused on measuring the model's performance using standard classification metrics: accuracy, precision, recall, and F1-score.

Accuracy measures the overall proportion of correct predictions, including both benign and malicious queries, out of all predictions made. *Precision* measures the proportion of queries the model flagged as malicious that was malicious, reflecting how accurate positive predictions are. *Recall* measures the proportion of all actual malicious queries that the model successfully identified, reflecting its ability to catch every threat and minimize missed detections. *F1-score* is the harmonic mean of precision and recall, providing a single metric that balances both false positives and false negatives to give a holistic view of model performance [1].

To demonstrate its effectiveness, the Improved Text-CNN was compared against several baseline models. The first was CNN, which uses filters to scan text for local patterns but does not use an attention mechanism to prioritize which patterns are most important.[10] Another baseline was the *Text-RNN*, a Recurrent Neural Network model that processes queries sequentially but is less focused on capturing local patterns [11]. The original *Text-CNN* model combined convolutional layers with word embeddings lacked the attention mechanism [12].

The testing results showed that the Improved Text-CNN model outperformed all these baselines, achieving the highest values across all metrics. It reached an accuracy of 92.40 percent, precision of 92.45 percent, recall of 92.29 percent, and an F1-score of 92.37 percent, indicating it was better at correctly identifying both obvious and hidden SQL injection attacks.

6.2 Results of XploitSQL

Testing XploitSQL involved measuring how well the system could generate SQL injection payloads that are both functional and evasive. The generated attacks were evaluated against various detection systems, including both open-source and commercial-grade Web Application Firewalls (WAFs) such as *ModSecurity* (an open-source WAF) and *Fortinet* (a commercial enterprise WAF) [5, 7]. Detection was also tested against machine learning classifiers like *CNNs*, *RNNs*, and *LSTMs*, which are deep learning models commonly used for text-based threat detection [10, 11].

The evaluation focused on three key criteria: *validity*, ensuring that generated payloads were executable SQL; *maliciousness*, verifying whether they successfully altered query logic; and *evasion*, determining whether the attacks could bypass detection systems [5]. Only valid and malicious queries were considered when calculating evasion rates.

The evasion rate was defined as the percentage of malicious queries that were not detected by a given defense system. For example, if 1,000 valid and malicious attack payloads were generated and 893 of them bypassed detection by a CNN model, the evasion rate would be (893 / 1000) × 100 = 89.3 percent.

XploitSQL was compared against three baseline systems: *AdvSQLi*, which generates adversarial SQLi through mutation rules; *WAF-A-MOLE*, a reinforcement learning–based evasion tool; and *LLM-SQLi*, which uses a large language model without reinforcement learning [2, 5, 9].

A key finding was the difference between the untuned large language model and the fully trained XploitSQL. Against a CNN-based detector, the base LLM-SQLi model achieved an evasion rate of 44.38 percent, while XploitSQL, after reinforcement learning, doubled its success to 89.37 percent.

Further testing against commercial WAFs showed similar results. Against ModSecurity, a widely used open-source detection tool, XploitSQL achieved a 63.58 percent evasion rate, outperforming AdvSQLi's 37.18 percent and WAF-A-MOLE's 49.10 percent. Against Fortinet, XploitSQL achieved 71.91 percent evasion. Even against Amazon's *AWS WAF* and *F5 Networks'* security platform, XploitSQL maintained its lead, achieving evasion rates over 65 percent in both cases [5].

7 Strengths and Weaknesses of Each Approach

The Improved Text-CNN model is a powerful deep learning system that excels at detecting both obvious and obfuscated SQL injection (SQLi) patterns. It learns semantic relationships through word embeddings, enabling it to generalize beyond simple keyword matching. Its convolutional layers effectively detect short, meaningful patterns (like "OR 1=1"), while the Channel Attention Mechanism (CAM) improves precision by focusing on the most relevant features. Testing showed high accuracy and F1-scores, confirming its strong performance against a variety of SQLi payloads, including those that evade traditional rule-based systems [10, 12].

However, Improved Text-CNN relies on learning only from the training data, limiting it to patterns seen during training, and may struggle with zero-day or novel attacks. Like many neural networks, it lacks interpretability, making it hard to explain why a query was flagged. Ultimately, it remains a passive defense—it reacts to attacks but does not anticipate or generate them.

XploitSQL, by contrast, represents an offensive application of machine learning. It combines a Large Language Model (LLM) with Reinforcement Learning (PPO) to generate syntactically valid, malicious, and evasive SQLi payloads. While it is designed to create attacks, XploitSQL also serves a defensive purpose by generating advanced adversarial examples that can be used to retrain and strengthen machine learning–based detection systems. Unlike mutation-based tools, XploitSQL adapts its strategies through feedback, consistently outperforming other generators across machine learning classifiers and commercial WAFs [5, 9].

Nonetheless, XploitSQL has limitations. It is computationally expensive to train, requires thousands of iterations, and lacks transparency into why a specific attack succeeds. Moreover, it raises ethical concerns, as such tools could be weaponized by hackers if not properly controlled.

8 Conclusion

The Improved Text-CNN and XploitSQL models represent two interconnected approaches to the SQL injection problem—one built for defense, the other for attack. Both use machine learning but differ in design and purpose.

Improved Text-CNN serves as a defensive classifier, using word embeddings, convolutional filters, and CAM to detect malicious SQL patterns with high accuracy and F1-scores, making it effective against known and slightly modified attacks. Machine learning models like this are needed because traditional rule-based systems often fail to catch mutated or hidden SQLi attempts.

XploitSQL, on the offensive side, combines LLMs with reinforcement learning (PPO) to generate evasive SQLi payloads. It achieved evasion rates up to 89.37 percent by continuously adapting its strategies against ML-based detectors and commercial WAFs.

Their interaction highlights a growing truth: defenses must evolve too. Static systems will fall behind, making adaptive, explainable machine learning models critical for the future of cybersecurity.

References

[1] Kemi Akanbi, Odunayo Gabriel Adepoju, and Kofi Isaac Nti. 2024. Developing A System for Automatic Prediction of Polycystic Ovary Syndrome Using Machine Learning. In *Proceedings of the 2024 7th International Conference on Machine Learning and Machine Intelligence*

(MLMI) (MLMI '24). Association for Computing Machinery, New York, NY, USA, 20–26. https://doi.org/10.1145/3696271.3696275

[2] Luca Demetrio, Andrea Valenza, Gabriele Costa, and Giovanni Lagorio. 2020. WAF-A-MoLE: evading web application firewalls through adversarial machine learning. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (Brno, Czech Republic) *(SAC '20)*. Association for Computing Machinery, New York, NY, USA, 1745–1752. https://doi.org/10.1145/3341105.3373962

[3] Jasmina Gajcin, Jovan Jeromela, and Ivana Dusparic. 2024. Semifactual Explanations for Reinforcement Learning. In *Proceedings of the 12th International Conference on Human-Agent Interaction* (Swansea, United Kingdom) *(HAI '24)*. Association for Computing Machinery, New York, NY, USA, 167–175. https://doi.org/10.1145/3687272.3688324

[4] Kritarth Jhala and Shukla Umang D. 2017. Tautology based Advanced SQL Injection Technique: A Peril to Web Application. In *Proceedings of the National Conference on Latest Trends in Networking and Cyber Security (NCLTNCS)*. IJIRST, Ahmedabad, India. https://www.ijirst.org/articles/SALLTNCSP008.pdf

[5] Daniel Leung, Omar Tsai, Kourosh Hashemi, Bardia Tayebi, and Mohammad A. Tayebi. 2024. XploitSQL: Advancing Adversarial SQL Injection Attack Generation with Language Models and Reinforcement Learning. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management* (Boise, ID, USA) *(CIKM '24)*. Association for Computing Machinery, New York, NY, USA, 4653–4660. https://doi.org/10.1145/3627673.3680102

[6] Muyang Liu, Ke Li, and Tao Chen. 2020. DeepSQLi: deep semantic learning for testing SQL injection. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis* (Virtual Event, USA) *(ISSTA 2020)*. Association for Computing Machinery, New York, NY, USA, 286–297. https://doi.org/10.1145/3395363.3397375

[7] G. Rama Koteswara Rao, R. Satya Prasad, and M. Ramesh. 2016. Neutralizing Cross-Site Scripting Attacks Using Open Source Technologies. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies* (Udaipur, India) *(ICTCS '16)*. Association for Computing Machinery, New York, NY, USA, Article 24, 6 pages. https://doi.org/10.1145/2905055.2905230

[8] Rajesh Sharma and Mia Tang. 2024. Machine Learning & Neural Networks. In *ACM SIGGRAPH 2024 Courses* (Denver, CO, USA) *(SIGGRAPH Courses '24)*. Association for Computing Machinery, New York, NY, USA, Article 14, 130 pages. https://doi.org/10.1145/3664475.3664574

[9] Chenyuan Yang, Zijie Zhao, and Lingming Zhang. 2025. KernelGPT: Enhanced Kernel Fuzzing via Large Language Models. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (Rotterdam, Netherlands) *(ASPLOS '25)*. Association for Computing Machinery, New York, NY, USA, 560–573. https://doi.org/10.1145/3676641.3716022

[10] Shuxin Yang, Min Guo, Jianqing Wu, Yishan Chen, Guangjian Huang, and Bowen Zeng. 2025. Hybrid Inverted Transformer-CNN Model for Train Primary-Delay Recovery Time Prediction. In *Proceedings of the 4th Asia-Pacific Artificial Intelligence and Big Data Forum (AIBDF '24)*. Association for Computing Machinery, New York, NY, USA, 1103–1110. https://doi.org/10.1145/3718491.3718669

[11] Jianyang Yu, Yuanyuan Qiao, Kewu Sun, Hao Zhang, and Jie Yang. 2018. Classification of Transaction Behavior in Tax Invoices Using Compositional CNN-RNN Model. In *Proceedings of the 2018 ACM International Joint Conference and 2018 International Symposium on Pervasive and Ubiquitous Computing and Wearable Computers* (Singapore, Singapore) *(UbiComp '18)*. Association for Computing Machinery, New York, NY, USA, 315–318. https://doi.org/10.1145/3267305.3267597

[12] Wei Zhao, Junling You, and Qinghui Chen. 2024. SQL Injection Attack Detection Based on Text-CNN. In *Proceedings of the 2024 International Conference on Generative Artificial Intelligence and Information Security* (Kuala Lumpur, Malaysia) *(GAIIS '24)*. Association for Computing Machinery, New York, NY, USA, 292–296. https://doi.org/10.1145/3665348.3665398