electronics

MDPI

Article

A Semantic Learning-Based SQL Injection Attack Detection Technology

Dongzhe Lu, Jinlong Fei * and Long Liu

State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China
* Correspondence: feijinlong_2021@163.com; Tel.: +86-13683818083

Abstract: Over the years, injection vulnerabilities have been at the top of the Open Web Application Security Project Top 10 and are one of the most damaging and widely exploited types of vulnerabilities against web applications. Structured Query Language (SQL) injection attack detection remains a challenging problem due to the heterogeneity of attack loads, the diversity of attack methods, and the variety of attack patterns. It has been demonstrated that no single model can guarantee adequate security to protect web applications, and it is crucial to develop an efficient and accurate model for SQL injection attack detection. In this paper, we propose synBERT, a semantic learning-based detection model that explicitly embeds the sentence-level semantic information from SQL statements into an embedding vector. The model learns representations that can be mapped to SQL syntax tree structures, as evidenced by visualization work. We gathered a wide range of datasets to assess the classification performance of the synBERT, and the results show that our approach outperforms previously proposed models. Even on brand-new, untrained models, accuracy can reach 90% or higher, indicating that the model has good generalization performance.

Keywords: SQL injection attack; deep learning; web application vulnerability detection

Citation: Lu, D.; Fei, J.; Liu, L. A Semantic Learning-Based SQL Injection Attack Detection Technology. *Electronics* **2023**, *12*, 1344. https://doi.org/10.3390/electronics12061344

Academic Editor: Rameez Asif

Received: 9 February 2023
Revised: 8 March 2023
Accepted: 11 March 2023
Published: 12 March 2023

1. Introduction

Injection vulnerabilities are specific vulnerabilities triggered by developers or imperfect design processes [1]. With the rapid expansion of the Internet infrastructure, the increasing reliance on digital information by an increasing number of users highlights the importance of information and data protection [2]. Data security is defined as the use of hardware or software to prevent unauthorized access, modification, or destruction of information.

Structured Query Language (SQL) injection is a code injection attack that executes the input data as code, thus violating the data-code separation principle [3]. An attacker can insert SQL commands into the query string of a web form submission, Uniform Resource Locator (URL), or page request and change the SQL statement execution logic to gain access to resources or change data stored in the database when a web application is passing SQL statements to the backend database for database operations without strict filtering of user input parameters, as shown in Figure 1.

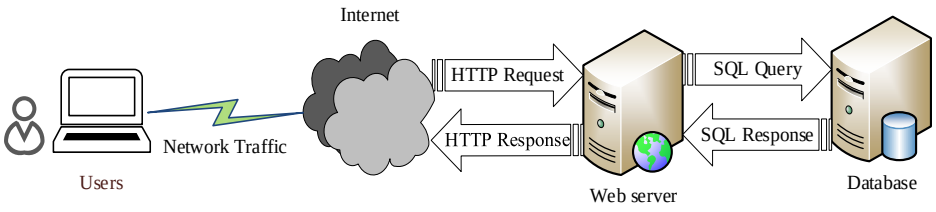

Figure 1. How SQL injection attacks work.

SQL injection attacks have become a popular tactic among cyber attackers due to their ease of implementation and high threat level. Traditional rule-based detection models are

inefficient and ineffective at detecting variant attacks. Simultaneously, there are numerous free SQL injection tools available on the Internet, lowering the bar for carrying out SQL injection attacks, while flaws in development languages, limitations in some developers' professionalism, and a lack of web security awareness increase the likelihood of successful attacks on websites. As a result, it is critical to develop a reliable and accurate SQL injection attack detection model for web application security.

In this paper, we present synBERT, a semantic learning and deep learning-based technique for detecting SQL injection attacks. For the first time, we apply a pre-trained model to the field of vulnerability attack detection and demonstrate that the model can fully learn sentence-level semantic information. It outperforms other detection algorithms in terms of classification accuracy and can distinguish between SQL injection statements and SQL statements. This paper's contributions are listed below.

(i) We collect a more comprehensive malicious dataset, covering the full range of SQL injection attacks. Moreover, benign samples are selected not only from the plain text but also from normal SQL statements. This method has the potential to reduce the number of false alarms.
(ii) For raw traffic, we detect injection attacks not only in URL fields but also in request body and request header fields. To some extent, this reduces the possibility of underreporting.
(iii) We propose a new detection model—synBERT, which is based on semantic comprehension. When compared with other detection models, it outperforms Convolutional Neural Networks (CNN) [1], Multilayer Perceptron (MLP) [4,5], Long Short-Term Memory (LSTM) [6,7], and others [8,9].
(iv) We use a structural probe to assess how well the synBERT model learns sentence-level semantic information, and we visualize it with heat maps and tree maps.

The remainder of the paper is organized as follows. Section 2 introduces SQL injection attacks and the most commonly used vectorization methods, as well as the benefits and drawbacks of traditional attack detection methods. Section 3 introduces our proposed model and its principles. Section 4 expands on the structural probe's design principles and visualization. Section 5 conducts a comprehensive test of synBERT's performance and selects a new test set for secondary evaluation. Section 6 concludes the paper and suggests future works.

2. Background and Related Work

2.1. SQL Injection Attacks

Structured Query Language Injection Attack (SQLIA) is one of the most important and dangerous vulnerabilities in interactive online applications [3]. Web applications with databases that store important information are one of the targets of SQLIA. According to Mitre [10], SQL injection attacks are one of the oldest, most prevalent, and most damaging types of security attacks facing web-based information systems.

The causes of SQL injection can be broadly grouped into two categories: dynamic string construction and insecure database configuration. The former includes incorrect handling of escape characters, incorrect handling of types, incorrect handling of join queries, incorrect handling of errors, and incorrect handling of multiple commits. The latter includes pre-installed users by default, running as root or SYSTEM or Administrator privileged system users, and allowing many system functions by default.

After summarizing, the classic SQL injection methods are classified in the following ways. According to the way of constructing and submitting SQL statements, they can be classified into: cookie injection, GET injection, POST injection, Hyper Text Transfer Protocol (HTTP) header injection, and second order injection. According to the execution effect, it can be divided into blind injection and backdated injection, mainly the following: boolean-based blind injection, time-based blind injection, error reporting-based blind injection, joint query injection, stacking injection, annotation character injection, wide-byte injection, and reduplication style.

In addition to this, SQL injection vulnerabilities are often exploited in more novel and sophisticated attacks, such as Fast-Flux SQL Injection and compound SQL Injection (a combination of a SQL injection attack and other web application attacks). There are many examples.

(a) SQL+ Distributed Denial of Service (DDoS): This attack is used to hang the server and exhaust the resources so that users cannot access it. To track the DDoS attack, the commands that can be used in SQL injection are encoding, compression, connection, etc.
(b) SQL+ Domain Name System (DNS) Hijacking: By using this type of attack, the attacker intends to embed a SQL query in the DNS request and capture it to make it propagate across the internet.
(c) SQL+ Cross Site Scripting (XSS): XSS is a client-side code injection attack where an attacker can inject malicious code into the input fields of an application. After inserting the XSS script, it will execute and try to connect to the application's database. The code to extract the data from the database can be obtained using the iframe command [11].
(d) SQL+ Insufficient authentication: if the security parameters are not initialized, an attacker can access sensitive content without verifying the user's identity. So attackers exploit this vulnerability to inject SQL injection code.

In general, the harms caused by SQL injection include the following. (a) An attacker can access the data in the database without authorization and steal the user's privacy as well as personal information, resulting in the leakage of the user's information. (b) Add or delete operations to the data in the database, such as privately adding or deleting administrator accounts. (c) If a website directory has write permission, the attackers can write a web page Trojan. The attacker can then tamper with the web page and publish some illegal information. (d) After steps such as privilege extraction, the highest privileges of the server are obtained by the attacker. The attacker can remotely take control of the server, install a backdoor, and be able to modify or control the operating system.

2.2. Traditional Detection Methods

Although the traditional vulnerability detection technology shows many shortcomings in many application scenarios, its research provides the technical basis and feasible development direction for the subsequent research of more intelligent vulnerability detection technology. The traditional methods for detecting SQL injection attacks are static analysis, dynamic analysis, combined static and dynamic analysis, and parameter filtering, and the detection architecture is shown in Figure 2.

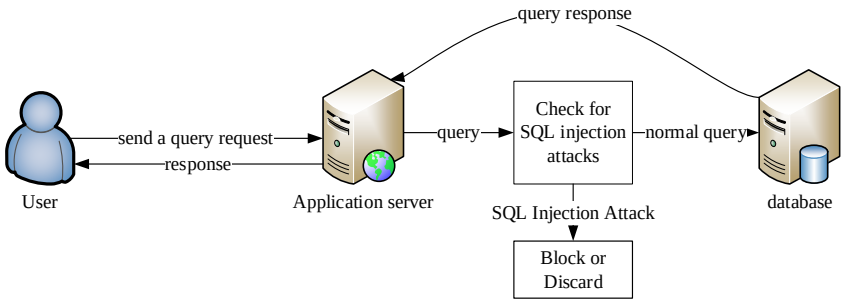

Figure 2. Traditional detection methods.

Static analysis methods detect type errors and syntax errors by analyzing incoming statements, which mainly refers to source code analysis without running the program to determine if there is a possibility of SQL injection. Many traditional web attack mitigation is achieved by statically analyzing incoming web traffic, also known as signature detection. This strategy involves creating a signature of a web attack, and when that signature is detected, suspicious traffic can be blocked through a firewall or other security device. This method has the advantage of being fast, implemented in real-time, and protecting network resources, but the disadvantage is that only known attacks can be detected.

Wassermann and Su [12] proposed a combined automated reasoning and static analysis approach to prevent SQL injection attacks and experimentally demonstrated that this approach can detect the presence of reduplicative attacks in real-time SQL queries. However, it cannot detect other types of attacks. A static analysis framework, called SAFELI [13], was proposed for identifying SQL injection attack vulnerabilities at compile time. SAFELI can identify vulnerabilities that may lead to violations of information security for the corresponding user input. The limitation of this approach is that SQL injection attacks are detected only in Microsoft-based products.

The dynamic analysis approach determines whether a statement is malicious by analyzing the final implementation of a statement, which mainly refers to detecting the presence of SQL injection attacks in an application by performing dynamic penetration tests at runtime or using generative models when the source code of the web application is not available.

Pan et al. [14] proposed a RSMT-based approach for web application attack detection that analyzes call tracing as the runtime behavior of a web application. RSMT uses an auto-encoder for analysis and uses graphs as data calls. In RSTM, a smaller amount of labeled data is used to calculate the reconstruction error of the autoencoder and to establish a threshold to distinguish between normal and abnormal behavior. However, further research is still needed to investigate the efficiency of retraining the autoencoder, and it may not be able to support the detection of zero-day attacks. Moreover, the authors did not compare with more complex network structures. Shin [15] proposed an automated white-box security testing framework to detect SQL injection attacks. The warnings generated by static analysis tools or automated black-box testing tools can be reduced, and the main limitation of dynamic analysis is that vulnerabilities in web applications can only be located by relying on the attacks that have already appeared. Therefore, this method can also not cope with zero-day attacks.

Combined static and dynamic analysis detection is a hybrid approach that combines static code inspection with real-time monitoring at runtime to detect and block SQL injection attacks. The static analysis phase analyzes the program source code and builds a SQL statement security model based on the program source code, while the dynamic analysis phase compares the requests submitted to the database with the model of static analysis and accesses them normally if they are the same or denies access if they are different.

Wang et al. [16] proposed a method for SQL vulnerability detection based on injection analysis techniques. Firstly, based on the combination of dynamic and static analysis, a detailed analysis of injection statements in terms of data flow and program behavior is carried out, and then a SQL vulnerability determination algorithm based on lexical feature comparison is implemented. Finally, a SQL vulnerability detection system is designed and built by combining alias analysis techniques, behavior models, and lexical feature comparison. ASSIST was a technique [17] that used a combination of static analysis and program transformation for automatic query sanitization to prevent SQL injection in code. However, similar to AMNESIA [18], this technique also leads to false negatives due to imprecision. Qing and He [19] proposed an approach to detect and prevent SQL injection attacks by using analytic techniques to extract legitimate SQL queries and match them with Aspect Oriented Programming (AOP). However, the main limitation of the method is that the source code of the program must be visible.

Parameter filtering methods mainly refer to filtering invalid characters based on regular expressions and blacklists. It is a hazy, non-independent concept that is frequently used in conjunction with static or dynamic analysis methods. Ref. [20] divides SQL parameters into six elements, uses the static analysis sequence of injection points and filtering omissions, and passes them as parameters to the dynamic detection module to generate more targeted test cases. However, there are some obvious drawbacks to this approach. One is that the test cases stitched together according to the rules are very limited, and the other is that the source code of the program must be known ahead of time. Katole et al. [21] combined query processing methods and parameter filtering methods to give a method to

detect the difference between the original SQL query and modified injected SQL query by parameters, which can detect SQL injection attacks before any data is lost from the web application, thus making the system more secure. The drawbacks are that detecting SQL injection attacks is time-consuming, and more robust methods need to be developed for different business requirements.

Traditional Web protection technology uses feature detection to detect SQL injection, and its method is to identify specific injection parameters at a specific access request URI and write thousands of detection rules depending on the type and version of the Web server where SQL injection vulnerabilities exist. However, these rules can only defend against SQL injection attacks on Web servers that have been discovered, and the detection features also lack universality, making it difficult to defend against more complex SQL injection and 0day injection attacks.

2.3. Artificial Intelligence-Based Detection Methods

AI detection methods for SQL injection first emerged using feature engineering plus machine learning methods, which use statistical feature extraction methods for feature engineering. It is followed by training various classifiers of machine learning or integrated learning models and detecting the presence of SQL injection attacks in real data streams. Statistical feature extraction includes feature extraction by word statistics and expert abstraction by empirical feature extraction methods. The former generally uses Term Frequency–Inverse Document Frequency (TF-IDF) or N-gram methods, and the latter relies on the experience of security experts. Under this approach, whether the data samples and expert experience are sufficient and effective determines the final detection results.

Makiou et al. [22] used a combination of a machine learning classifier and rule engine to detect SQL injection attacks, improving the detection performance of traditional rule-based methods. First, the HTTP request traffic is divided into different parts to detect the presence of SQL injection. A plain Bayesian classifier is used to characterize the presence of certain keywords and symbols in SQL statements. To reduce false positives during training, the model gives a higher penalty for misclassifying SQL injection attacks as normal traffic. After the classification is complete, the suspicious traffic is forwarded to a pattern-matching detection engine based on a simplified security rule set for in-depth analysis. The accuracy of the method is 97.6%. The advantage is that the detection location contains not only URLs and POSTs but also HTTP headers, whereas most detection methods ignore SQL injection detection in headers for simplicity and efficiency purposes. The disadvantage is that the analysis continues using the rule engine after the machine learning model, which loses the generalization of machine learning and does not overcome the limitations of the rules themselves to detect unknown SQL attacks.

Choi et al. [23] introduced general methods of natural language processing, such as the n-gram, into the detection of SQL injection statements with good results. Candidate tokens for the n-grams were limited to SQL functions pulled from an ordered occurrence frequency list in SQL Command. However, the method cannot cope with complex joint injection attacks. Obviously, due to the limitations of the n-gram, the method is also unable to cope with multiple obfuscated and mutated injection statements. A new approach for SQLi detection based on unsupervised machine learning [24] used a dual Hidden Markov Model (HMM), but the process was very complex and time-consuming, which affected the performance and resulted in more false positives. In [25], the authors made further improvements by developing methods that are more suitable for fast traffic networks and SQL injection attack detection. The authors proposed the use of machine learning methods for detection, providing an accurate means of improving network egress filtering.

Natural language processing based on deep learning has received increasing attention in recent years and has made significant advances, such as the use of convolutional neural networks for text classification to better capture the local relevance of text. Deep learning, unlike traditional machine learning algorithms, does not require complex feature engineering, and its performance typically improves with increasing training data. In

other words, it is far more scalable than traditional machine learning algorithms. A wide range of researchers [26] continue to propose new techniques in the field of SQL injection attack detection, and detection performance has improved as a result. Of course, no single technique can solve all problems, and each has its own set of scenarios to consider as well as drawbacks.

Fang et al. [27] designed to convert sample data into word vectors as the input of the model and uses LSTM models to detect, demonstrating that LSTM-based methods have better detection results than Recurrent Neural Network (RNN) and CNN-based methods. However, there are still many challenges and problems. First, due to the lack of open SQL injection datasets, it is difficult to obtain enough training samples during the training process, leading to overfitting. Second, there are many types of SQL injection attacks, each with its own specific performance and judgment features, and the samples of different forms of attacks are unevenly distributed. When the distribution is unbalanced, i.e., the training data for a certain type of anomaly is too small, it is difficult for deep learning-based methods to achieve good detection results.

Ref. [8] built the MLP and LSTM models to learn the statistical features and character order features of URL parameter strings, respectively. The experimental results show that the accuracy of the MLP model is much higher than that of the Naïve-Bayes algorithm, the SVM algorithm, the SVM + Social Media Optimization (SMO) algorithm, and the SVM + Particle Swarm Optimization (PSO) algorithm proposed by some scholars. The LSTM model, although its detection time is much longer than that of the MLP model, it has a unique advantage. On the contrary, the effect of this statistical feature extraction is very sample-dependent, i.e., the model trained in this case has a weak generalizability, which will be mentioned later. Zhongdong Zhu [4] systematically proposed a framework for detecting SQL injection attacks based on HTTP traffic in the context of complex HTTP traffic. The model building module proposed a modeling method for detecting payloads of arbitrary length and a training method for variable-length sequences with guaranteed efficiency. The framework had a low false alarm and missing reporting rate in a real network environment. However, the author only proposed a theoretical framework without providing any experimental data or results. Moreover, the method of concatenating statistical features and word embeddings is theoretically ineffective because the dimension of word embeddings is dozens or even hundreds of times that of statistical features.

Currently, the latest machine learning approaches in Web security focus more on methods that combine machine learning algorithms with semantic features of Web code, caring not only about the statistical features of words in a sample but also about the contextual situational semantics of words, achieving better practical results than traditional detection methods and statistical-based AI approaches.

Muyang Liu et al. [28] proposed a deep natural language processing-based tool Deepsqli for generating test cases for detecting SQL injection vulnerabilities. By employing deep learning-based neural language models and lexical prediction sequences, Deepsqli can learn the semantic knowledge embedded in SQL injection attacks, enabling it to transform user input (or test cases) into a semantically relevant and potentially more complex new test case. Experiments were conducted to compare Deepsqli and Sqlmap on six real-world web applications with different scales, features, and domains. The empirical results demonstrate the effectiveness and superiority of Deepsqli, leading to the identification of additional SQL injection vulnerabilities. This approach can be used to exploit unknown SQL injection vulnerabilities, but it still gives us some insights, especially in the areas of test case generation and semantic knowledge learning. Ref. [29] proposed a model for SQL injection attack detection based on machine learning techniques implemented in the business logic layer of Web applications. It claimed to improve the efficiency of SQL injection attack detection by extracting semantic features from dynamic and static analysis that could effectively indicate SQL injection attacks. However, this work is only theoretical and has not been tested by experiments.

3. Detection Model synBERT

It is well known that Bidirectional Encoder Representations from Transformers (BERT) are used for pre-training to learn word vector representations of specific words in a given context. BERT uses the encoder part (left of Figure 3) of the transformer architecture [30]. Its most important feature is that it discards traditional RNNs and CNNs and effectively solves the tricky long-term dependency problem by transforming the distance of two words at arbitrary positions through an attention mechanism.

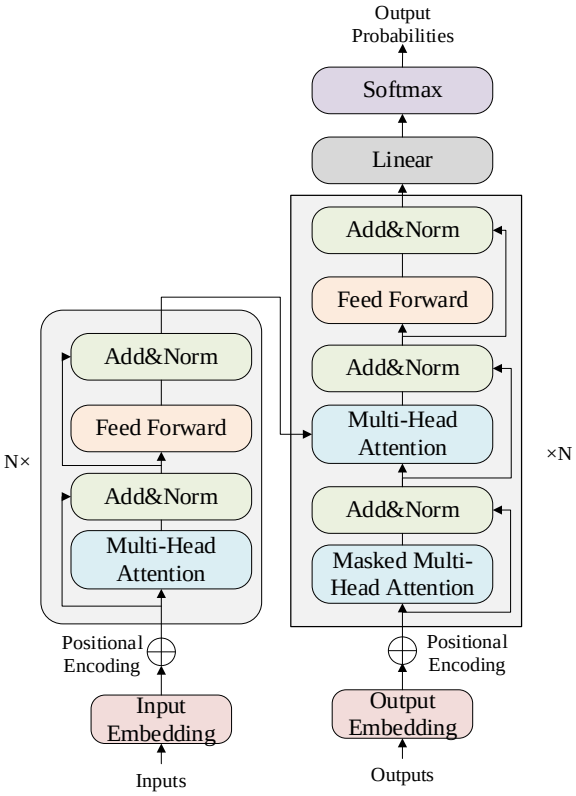

Figure 3. The architecture of transformer.

As shown in Figure 3, the gray area on the left is a block, and each block contains multiple layers. Add & Norm adds the input and output vectors of the multi-head attention layer to get a new set of vectors and then performs layer normalization. The linear layer is a fully connected neural network, which projects the vectors generated by the encoder as logits. Then the softmax layer normalizes the linear layer's output to get log_probs.

The input of the BERT encoder is a sequence of tokens, and the output is a sequence of vectors, where each input token corresponds to a vector. The input of the BERT model has three parts: token embedding, segment embedding, and position embedding. The internal structure of BERT consists of two parts. First, an initial embedding is created for each token by combining a pre-trained word vector with a position vector and text vector information. The position encoding is independent of the input content. Furthermore, this initial embedding sequence is passed through multiple blocks to produce a new sequence of contextual embeddings at each step.

In this paper, synBERT is designed for the features of SQL injection statements, with three component modules: the knowledge layer, the representation layer, and the encoder layer. The knowledge layer is responsible for the injection of knowledge and the generation of sentence syntax trees, the representation layer is responsible for projecting the syntax tree, a sequence of inputs, into a representation vector; and the encoder layer is responsible for encoding the representation vector based on the information provided by the injected triadic knowledge.

In this paper, the original position embedding is changed to a semantically relevant position embedding, i.e., the semantic information of the SQL statement is embedded explicitly in the input layer. Figure 4 visually represents how we replace the original position embedding with a custom embedding.

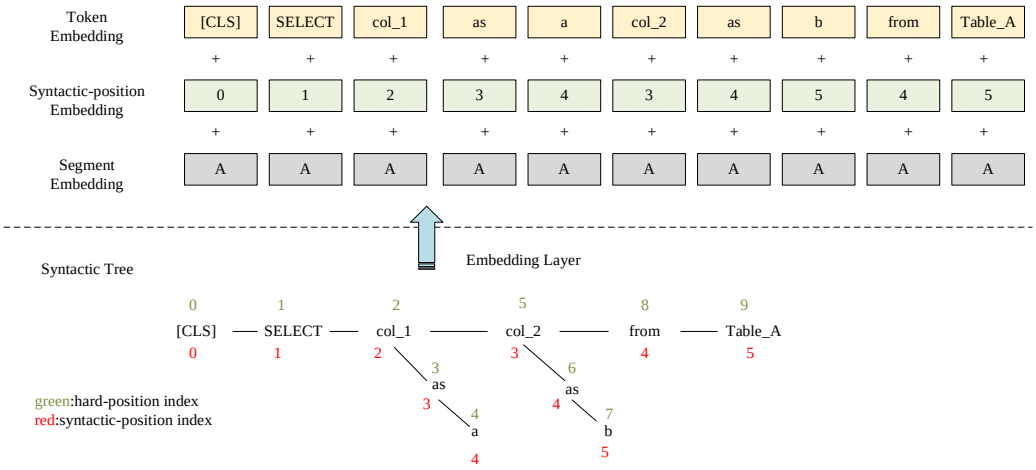

Figure 4. Three embeddings of the input.

synBERT describes the three embedding features of the input as wordpiece embedding, syntactic-position embedding, and segment embedding. Wordpiece refers to the division of words into a limited set of common subword units that can strike a compromise between word validity and character flexibility. Syntactic-position embedding refers to encoding the semantic and syntactic information corresponding to each token into a feature vector. Segment embeddings are used to distinguish between two sentences, e.g., whether B is the continuation of A. For sentence pairs, the feature value of the first sentence is 0 and the feature value of the second sentence is 1.

Therefore, we use two submodels for the task of computing and classifying the embedding vectors. First is a preprocessor, which converts a string into a numeric tensor. The second one is an encoder that accepts the tensor dictionary returned by the preprocessor and performs the trainable part of the embedding vector computation. This split processing allows the input to be processed asynchronously before being fed to the training loop. Figure 5 illustrates the structure of the synBERT model.

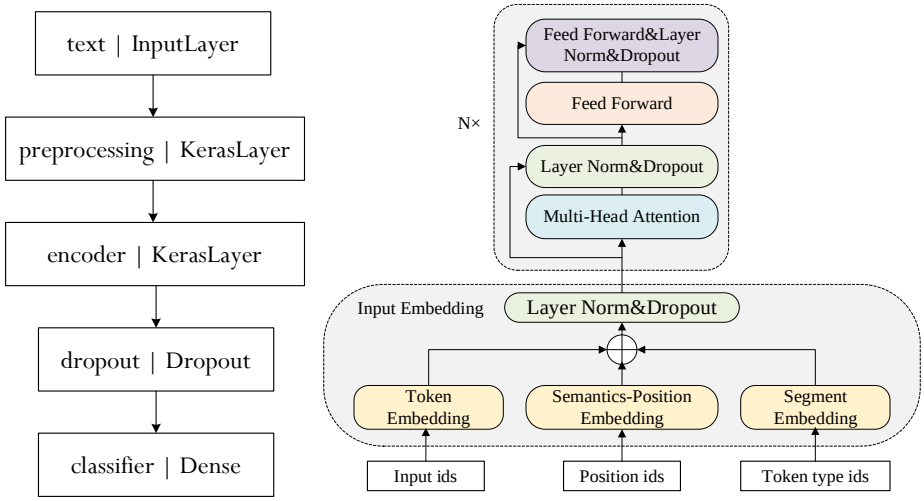

Figure 5. The structure of synBERT.

At the same time, we scale down the original model to reduce the number of neurons, resulting in a significant reduction in the amount of machine operation. Compared with the baseline model of BERT, synBERT is able to increase the computational speed by more than five times. The specific parameter values and meanings are described in Section 5.1.

4. Structure Probe

4.1. Syntax Tree of SQL Statements

For SQL statement compilation, lexical analysis is used to split the SQL string into tokens that contain keyword recognition. Take the statement "select a from table where c = 'aaaaaaaaa'" as an example, and split it into words to get [select, a, from, table, where, c, =, 'aaaaaaaaa'], each of which has a different attribute. That is to say, "select" is a DML statement, while "=" is a comparison operation symbol. Syntax analysis involves the use of top-down or bottom-up algorithms to parse tokens into abstract syntax trees. Syntax parsing rules for SQL statements, generally consist of two steps: identifying syntax-related statements and adjusting the syntax tree structure. Take the statement "Select col_1, col_2 as b from DT_A;" as an example, the level of its syntax tree is as follows.

```
|- 0 DML 'Select'
|- 1 Whitespace ' '
|- 2 IdentifierList 'col_1, ... '
|- 0 Identifier 'col_1'
|  '- 0 Name 'col_1'
|- 1 Punctuation ','
|- 2 Whitespace ' '
|  '- 3 Identifier 'col_2, ... '
|- 0 Name 'col_2'
|- 1 Whitespace ' '
|- 2 Keyword 'as'
|- 3 Whitespace ' '
|  '- 4 Identifier 'b'
|  '- 0 Name 'b'
|- 3 Whitespace ' '
|- 4 Keyword 'from'
|- 5 Whitespace ' '
|- 6 Identifier 'DT_A'
|  '- 0 Name ' DT_A'
'- 7 Punctuation ';'
```

The objects in the above syntax tree structure are inherited from the base class Token, and the parent node contains the information of the child nodes. Token has two instance attributes ttype and value, and the class attributes and the regular expressions used for parsing are shown in Table 1.

Table 1. Class attributes and their regular expressions.

Base Types and Sub-Attributes	Examples	Example of Regular Expressions
Comment	–	–|)#\+.*?(\r\n|\r|\n|$
Assignment	var:=val	:=
Operator	+	[+/@#%^&|^-]+
Comparison	=, >	[<>=~!]+
Text	\r\n	".*?"
Whitespace		\s
Newline	\r	\r\n|\r|\n
Punctuation	; : ( ) [ ] , .	[;:()\ [\ ],\ ...]

Table 1. *Cont.*

Base Types and Sub-Attributes	Examples	Example of Regular Expressions
Keyword	from, GROUP BY	CASE | IN | VALUES | USING | FROM | GROUP\s+BY
DDL	CREATE, ALTER	CREATE(\s+OR\s+REPLACE)?\b
DML	SELECT, UPDATE	SELECT | INSERT | UPDATE | DELETE
Name	c	(?<![\w\])])(\ [[^\ ]\ []+\ ])
Placeholder	?, *	(?<!\w)[$:?]\w+
Literal	hello	\d+ | \w+ | .+
String	aaaa	'\'.*?\'
Number	111	\d
Identifier	database, table, column	\w
Wildcard	*	*

Note: The first column shows all the attributes and sub-attributes that "Token" contains. The content in each solid line box indicates the same class. The dotted lines make the table more readable. Using "Text" as an example, the inclusion relationship (from lowest to highest) is as follows: Newline -> Whitespace -> Text -> Token.

4.2. Structure Probe for synBERT

Sentences are sequences of discrete symbols, but neural networks operate on continuous data–vectors–in high-dimensional space, and it is clear that a successful network will transform the discretized input into some sort of geometric representation. For example, the influential Word2Vec system [31], has been shown to place related words near each other in a space where certain directions in the space correspond to semantic distinctions. Syntactic information, such as numbers and tenses, is also represented by the orientation of space. An analysis of the internal states of RNN-based models shows that they represent information about soft-level grammars in a form that can be extracted by a hidden layer network [32]. Qualitative, visualization-based work [33] suggests that attention matrices may encode important relationships between words. One finding of Hewitt and Manning [34] was that BERT appeared to create a straightforward representation of the entire parse tree, which motivated our work. The authors found that the square of the Euclidean distance between contextual embeddings is roughly proportional to the tree distance in dependency parsing. Ref. [35] clarified the internal representation of linguistic information by BERT, finding evidence of syntactic representations in the attention matrix. The papers cited above show that language processing networks create a rich set of intermediate representations of semantic and syntactic information.

In this paper, we use the structural probe to verify whether the improved synBERT model reflects the syntactic tree structure of SQL statements, i.e., we can prove that the BERT model learns the syntactic tree structure as long as the distance between word vectors exhibits the pattern of the Pythagorean theorem.

Assuming that the two nodes of the syntax tree are w_i and w_j, two words for h_i and h_j, computing the distance between the two high-dimensional vectors can be transformed into

$$d = (h_i - h_j)^T (h_i - h_j) \tag{1}$$

This can be translated into computing a semi-positive definite measure matrix $A = B^T B$ to satisfy the

$$d_B(h_i, h_j) = (h_i - h_j)^T A (h_i - h_j) = [B(h_i - h_j)]^T B(h_i - h_j) \tag{2}$$

The matrix elements of the B-matrix are learned by training a neural network with a loss function set to the difference of the distance between the word vectors $d_B(h_i, h_j)$ and the word tree distance $d_{tree}(w_i, w_j)$, i.e.,

$$loss = \sum_l \frac{1}{|s^l|^2} \sum_{i,j} \left| d_{tree^l}\left(w_i^l, w_j^l\right) - d_B\left(h_i^l, h_i^l\right)^2 \right| \quad (3)$$

where l denotes any statement, $\left|s^l\right|$ denotes the number of words in the lth statement.

We use the trained synBERT to perform dependency parsing tests on SQL statements, and the testing Algorithm 1 is shown below.

Algorithm 1: Structure Probe Resolution Dependency Algorithm

Input: Sentence of n words $w_{1:n}^l$, a sequence of vector representations $h_{1:n}^l$, a linear transformation $B \in R^{k*m}$, a positive semidefinite symmetric matrix $A \in S_+^{m*m}$
Output: parse depth $||w_i||$
Steps:

(1) **for** i **do**
(2) $d_B(h_i) \leftarrow [B(h_i)]^T B(h_i)$
(3) $h_{iA} = d_B(h_i)$
(4) return torch of shape $(batch\_size, max\_seq\_len)$
(5) **for** i, j **do**
(6) $d_B\left(h_i, h_j\right) \leftarrow \left[B\left(h_i - h_j\right)\right]^T B\left(h_i - h_j\right)$
(7) $||\left(h_i - h_j\right)||_A = d_B\left(h_i, h_j\right)$
(8) return torch of shape $(batch\_size, max\_seq\_len, max\_seq\_len)$
(9) UnionFind(matrix(d_B))
(10) return a single distance image and depth image

Taking the statement "Select col_1, col_2 as b from Table_A;" (Note: Table_A here is the table name of the example, not a chart in this paper) as an example, the visualization of the parsing depth and parsing distance obtained using the structure probe is shown in Figure 6. It can be seen that the results are generally similar to those of the SQL parser: "b" has the deepest depth in the syntax tree, followed by "as"; "col_2" is similar to "col_1"; "Table_A" is deeper than "from". The heat map of the predicted parsing L2 distances is shown in Figure 7.

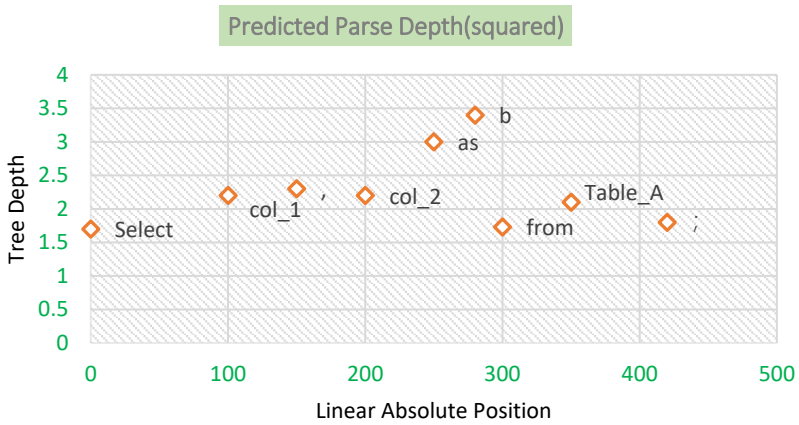

Figure 6. Predicted parse depth. The horizontal coordinate is the position of the word in the SQL statement, and the vertical coordinate is the depth (relative value) of the word in the syntax tree learned by the model. It can be seen that the depth of the syntax tree computed by the synBERT model is similar to the actual value.

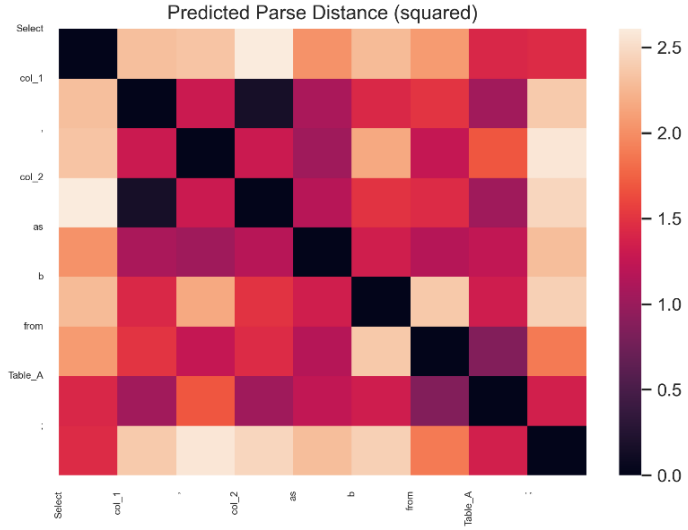

Figure 7. Predicted parse distance. This heat map shows the distance between two of these tokens. The light to dark color indicates the distance from near to far. For example, it can be seen that "col_1" and "col_2" are closest together, and they are at the same depth in the syntax tree.

The results are mapped to the distance of each token in the parse tree, and tikz-dependent LaTeX is returned, exporting its constructed minimum spanning tree in PDF format, as shown in Figure 8.

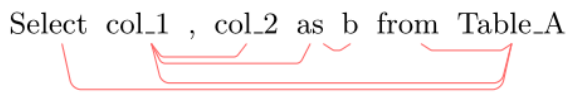

Figure 8. Minimum spanning tree exported by LaTeX.

Experiments show that using the synBERT enables lexical-level semantic analysis and sentence-level semantic analysis of SQL statements and SQLi statements. Furthermore, more accurate detection and classification of SQL injection attack statements can be made.

5. Experimental Results and Analysis

5.1. Evaluation Indicators

The evaluation metrics proposed in this paper aim to evaluate the detection model in a dichotomous scenario. We used the following evaluation metrics: accuracy, true positives, false positives, true negatives, and false negatives.

According to the confusion matrix shown in Table 2, the accuracy represents the proportion of the total sample that the model predicts correctly and is calculated as

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{4}$$

where TP is a correctly classified SQLi sample, FP is a non-injected sample incorrectly classified as SQLi, TN is a correctly classified non-injected sample, and FN is a SQLi sample incorrectly classified as normal.

Table 2. Confusion matrix.

		Predicted	
		Positive	**Negative**
Actual	**Positive**	TP	FN
	Negative	FP	TN

Recall, also known as TPR, indicates the percentage of positive samples correctly predicted for all actually positive samples. The precision rate indicates the percentage of positive samples that are correctly predicted for all samples that are predicted positive. The precision and recall rates are calculated as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{6}$$

F1 score combines Precision and Recall and is a summed average of the two.

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}} \tag{7}$$

The FPR, or false positive rate, reflects the ability of the model to correctly predict the purity of positive samples.

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}} = \frac{\text{FP}}{\text{N}} \tag{8}$$

5.2. Dataset Evaluation

The positive sample dataset in this paper is derived partly from public channels, namely Common Vulnerabilities & Exposures (CVE), China National Vulnerability Database (CNVD), Exploit Database, and other public vulnerability information repositories within the last four years, and partly from vulnerability recurrence and contest collection. The negative sample dataset is derived from network traffic on campus and the Kaggle website. This covers four major scenarios. We build the vulnerability environment based on the docker-compose dockerfile and capture HTTP traffic for vulnerability information provided by websites (e.g., VULHUB). We write programs to extract the attack payloads from samples that exist as traffic packets (mostly obtained through contests). We manually analyze and extract samples that exist in log form. Samples that are valid attack payloads can be used directly. For the unknown samples, we extract all available fields. This includes the query string for HTTP GET requests, the request body for POST requests, the User-Agent, Referer, Cookie, and X-Forwarded-For. The collected samples, according to our statistics, contain a variety of SQL injection attacks, including boolean-based blind injection, time-based blind injection, error-based blind injection, stacked injection, wide-byte injection, union queries, and tautologies. We also consider the dialects of various databases, such as MySQL, Oracle, SQL Server, and PostgreSQL. Such samples, however, are in the minority, as attackers frequently use standard SQL to try to inject without knowing what database the website is using. The statistical information of various database samples is shown in Table 3.

Table 3. Statistical information of various database samples.

Type	Malicious	Benign
MySQL	896	3155
Oracle	1035	
SQL Server	957	
PostgreSQL	48	
Plain Text	/	
Total	2936	3155

According to previous research, almost all results emphasize data preprocessing and feature engineering methods to improve model performance. The data preprocessing

methods that correspond to different feature engineering methods may differ, and we only discuss the generalized preprocessing methods here. For the positive samples, we first perform URL decoding, UNICODE decoding, and BASE64 decoding in order, as shown in Table 4. Following decoding, the normalization process is carried out, which includes converting all uppercase letters to lowercase letters, using "datetime" for strings that conform to date format, using 0 for integers, and rewriting keywords to remove redundancy.

Table 4. Decoding methods and examples.

Coding Methods	Before Decoding	After Decoding
URL decoding	?id=1%20union%20Select%201,2,group_concat(concat(username,0x7e,password))%20from%20iwebsec.users%0A1%0A	?id=1 union Select 1,2,group_concat(concat(username,0x7e,password)) from iwebsec.users
UNICODE decoding	se%u006cect	select
BASE64 decoding	4oCdIHVuaW9uIHNlbGVjdCAxLDIsZGF0YWJhc2UoKSM=	union select 1,2,database()#

The model used in this paper is synBERT-uncased, i.e., all tokens are converted to lowercase at the time of preprocessing. The parameters used for training are shown in Table 5.

Table 5. The parameters of synBERT.

Parameters	Parameter Meaning	Parameter Value
num_hidden_size	number of hidden layer neurons	256
num_hidden_layers	number of hidden layers in Transformer encoder	2
num_attention_heads	number of heads in multi-head attention	4
hidden_func	hidden layer activation function	gelu
hidden_dropout	hidden layer dropout rate	0.1
attention_dropout	dropout rate of the attention	0.1
epoch	number of training epochs	10

By adjusting some hyperparameters to make the model perform better and plotting the accuracy and loss curves with epochs for the training and validation sets. It can be seen from Figure 9 that the loss function of the model converges to an accuracy of about 0.9974 and a loss of about 0.0371, and reaches an accuracy of 0.9973 and an F1 score of 0.9964 on the validation set. We will then compare the methods with those of other papers.

The main controlled experiments in this section are from Refs. [1,4,5,7]. Since the code and datasets used in these papers are not open source, the model structure and parameter settings are first reproduced as described in these papers. Preliminary experiments show that the model does not fully achieve the performance metrics described in the original papers after changing the dataset, so we fine-tune it to make the comparison results more convincing.

The feature vector methods used in the comparison experiments include the statistical feature vector method, the TF-IDF vector method, and the word2vec method. The classification algorithms or models used are: CNN, LSTM, and MLP. (These algorithms and models are derived from the aforementioned literature.)

It should be noted that the statistical feature vectors are highly relevant to the dataset. In this paper, the following features were identified by data analysis of the existing dataset in conjunction with the findings Refs. [36,37], as shown in Table 6.

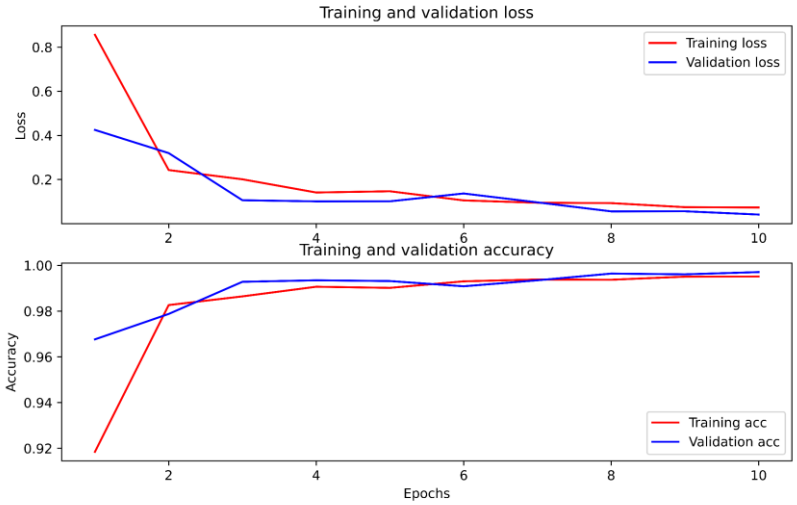

Figure 9. Accuracy and loss curves.

Table 6. Selected features and meanings.

	Feature	Meaning
1	query_len	Length of each query
2	num_word_query	The total number of words in a query
3	no_single_qts	The number of single quotes in the query
4	no_double_qts	The number of double quotes in the query
5	no_punct	Total number of punctuation marks in a query
6	no_single_cmnt	Number of single line comments in the query
7	no_double_cmnt	Number of multi-line comments in the query
8	no_white_space	The number of spaces in the query
9	no_percent	Number of percent signs
10	no_log_optr	Total number of logical operators in the query
11	no_arith_oprtr	Total number of arithmetic operators
12	no_null_val	Total number of null values in a query
13	no_hexdec_val	Total number of hexadecimal values
14	no_alphabet	Total number of letters in a query
15	no_digits	Total number of digits
16	len_of_chr_char_nul	Total number of chr + char + null keywords
17	genuine_keywords	Total number of keywords select, top, order, fetch, join, avg, count, sum, rows, etc.

By plotting the relationship between individual features and labels, we check whether the feature contributes to the prediction output. Distribution, box-line, violin, and Q-Q plots of log-transformed distributions and Box-Cox-transformed Q-Q plots are plotted, respectively. The features with higher discrimination, i.e., 3 to 17, are selected. As shown in Figures 10–13, the distribution and Q-Q plots on feature 8, it can be seen that the query has a high probability of becoming a positive sample as the number of spaces increases to 25 afterward. However, neither the log-transformed nor the power-transformed distribution plots show a good Gaussian distribution shape, so the feature is not taken. Furthermore, pairwise plots of the sampled features were plotted, and bivariate analysis was performed to find the range of features that would help distinguish them. Figure 14 shows the paired analysis plot for features 7 to 10. It can be obtained that features 3 and 6, features 5 and

6, features 11 and 15, and features 12 and 15 are the features that help predict the labels. Finally, t-distributed Stochastic Neighbor Embedding(t-SNE) plots are drawn to analyze whether the features work. As in Figure 15, the vast majority of points are well separated.

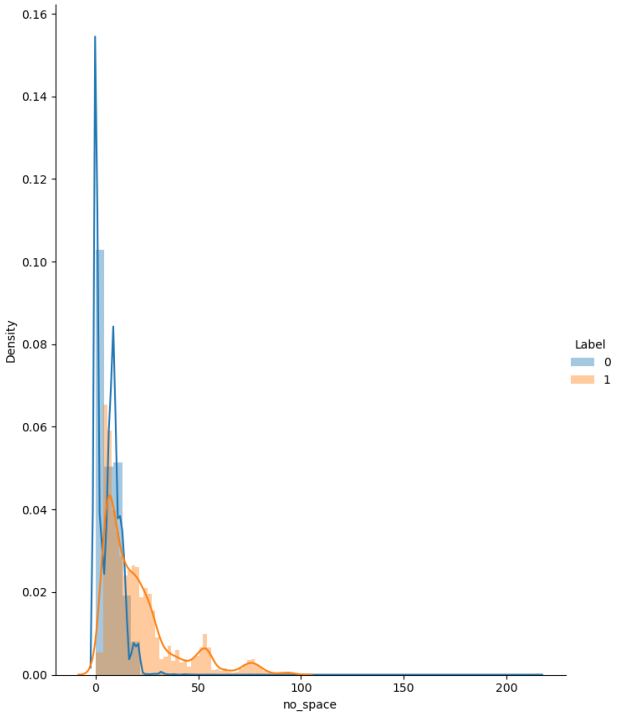

Figure 10. The distribution of feature 8. The horizontal coordinate indicates the number of spaces in the SQL query, and the vertical coordinate indicates the corresponding probability density function. The blue line indicates benign samples and the orange line indicates malicious samples. As can be seen, queries with the number of spaces after 25 have a high probability of becoming malicious.

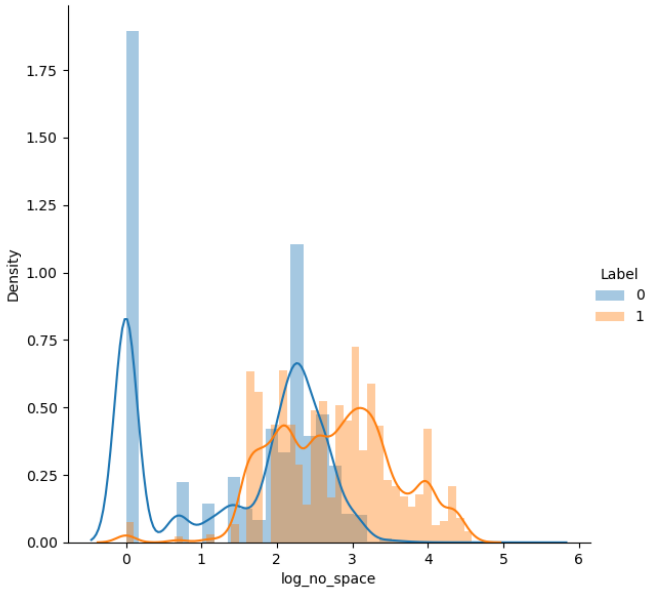

Figure 11. Log-transformed distribution of feature 8. The horizontal coordinate indicates the logarithm of the number of spaces in the SQL query, and the vertical coordinate indicates the corresponding probability density function. The blue line indicates benign samples and the orange line indicates malicious samples. As can be seen, this distribution is not a perfect Gaussian distribution and there is a lot of overlap between the two categories.

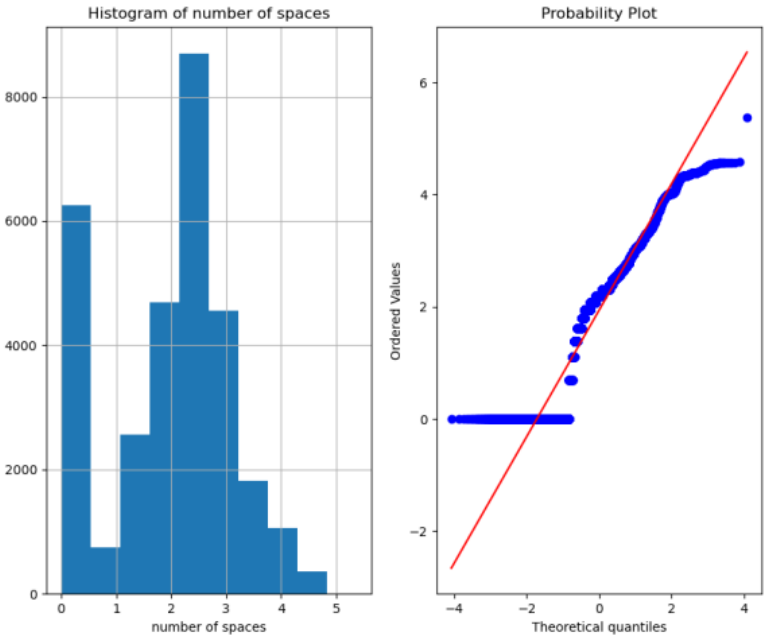

Figure 12. Q-Q plot of feature 8 by log transformation. After doing Box-Cox transformation on the distribution of the number of spaces, the sample frequency histogram (**left**) and Q-Q plot (**right**) are plotted. In the Q-Q plot (right), the horizontal coordinate indicates the normal distribution quantile and the vertical coordinate indicates the data quantile. It is obvious that the feature of the number of spaces does not follow a normal distribution.

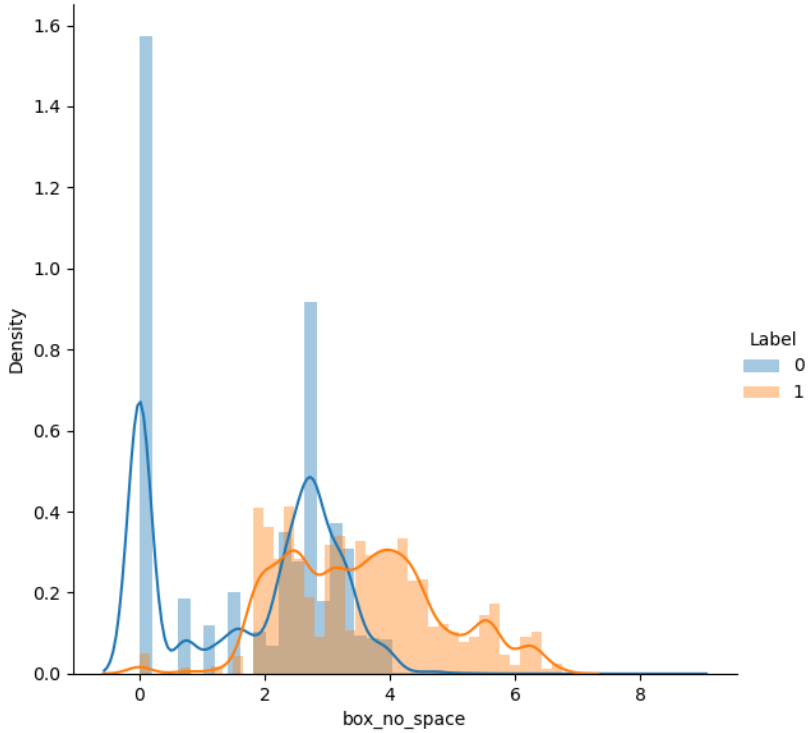

Figure 13. Distribution map of feature 8 after Box-Cox transformation. The horizontal coordinate indicates the Box-Cox transformation of the number of spaces in the queries, and the vertical coordinate indicates the corresponding probability density function. The blue line indicates benign samples and the orange line indicates malicious samples. It can be seen that there is still plenty of overlap between the two categories.

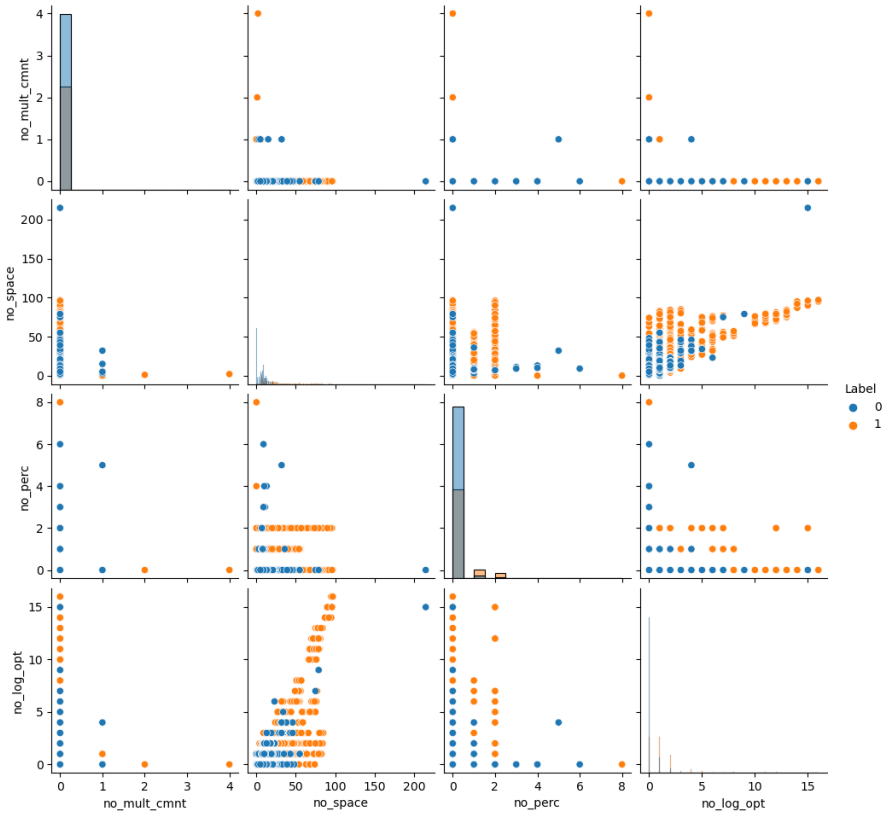

Figure 14. Paired plots of sampled features. The correlation between the two variables was determined by selecting features (here, features 7 to 10) by random sampling and performing bivariate analysis.

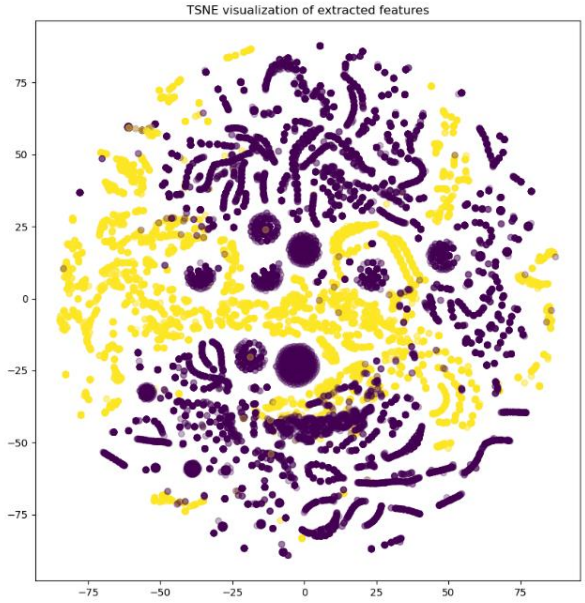

Figure 15. T-SNE dimensionality reduction classification result. The available features selected by the bivariate analysis were used to plot the reduced dimensional t-SNE plot. As can be seen, the vast majority of samples are well separated, indicating that these features are useful.

Different preprocessing and embedding approaches and different classification algorithms are used to binary classify. Table 7 shows the metrics for all comparison experiments.

The quantitative results show that the embedding encoding using synBERT has better characterization than others.

Table 7. The experimental results.

Experimental Methods	Accuracy (%)	Precision (%)	Recall (%)	F1 (%)	FPR (%)
Statistical features + MLP_1	87.84	90.63	84.01	87.19	13.06
Statistical features + $LSTM_1$	81.47	82.56	80.80	81.67	17.45
Statistical features + CNN_1	88.83	88.32	85.78	87.03	10.24
TF-IDF + MLP_2	89.37	85.53	71.09	77.64	10.12
TF-IDF + $LSTM_2$	92.11	85.03	73.40	78.79	8.01
TF-IDF + CNN_2	93.16	82.56	80.80	81.67	6.98
Word2Vec + MLP_3	92.82	90.67	92.91	91.78	7.52
Word2Vec + $LSTM_3$	93.14	93.56	92.43	92.99	6.40
Word2Vec + CNN_3	96.10	97.28	99.11	98.18	3.83
Method of our paper	99.74	99.68	99.52	99.60	0.56

Note: The subscripts 1, 2, and 3 are used to distinguish between model structures that are not identical.

Using statistical features to do detection, the average detection accuracy is 86.04%, and F1 is 85.29%, which cannot achieve high metrics. This is due to the variability of SQL injection samples. If the attacker deliberately changes the statistical features using obfuscated attack samples, this method will have missed and false positives. For SQLi statements, words that occur more frequently are not necessarily important. The TF-IDF method (average detection accuracy is 91.54% and F1 is 79.36%) does not assign the weights of feature words well and will have the problem of insufficient feature extraction. In contrast, word2vec, as a typical vectorization method, is able to map the distance of words to the distance in space. Its disadvantage is that, as a static vector, it does not take into account the discourse order problem. The word2vec method (average detection accuracy is 94.02% and F1 is 94.31%) is limited by the size of the window and cannot consider the relevance of a token in the whole sentence. Thus, the definition of semantic similarity in word2vec is, to some extent, not necessarily related to common sense. The BERT model only utilizes common contextual features such as word embeddings, and they rarely consider structured semantic information. However, the synBERT algorithm improves on these drawbacks. It can embed semantic information in SQL statements to provide rich semantics for language representation.

5.3. Generalizability Testing

We chose a totally new test set to evaluate the generalization ability of the final model formed by multiple algorithms, but not as a basis for the choice of tuning model parameters. The data for the test set is derived from the sample information of SQL injection attacks published in the 2022 CVE repository as well as normal SQL statements, with no crossover between the training and test sets, to further validate the generalization performance of the above-trained models. Table 8 shows the experimental results.

As time goes by, the game between attackers and defenders gradually escalates. More and more obfuscation and mutation mean traditional detection methods are no longer applicable, and the advantages of artificial intelligence are gradually coming to the fore. Semantic-based representations can help machines better identify attack traffic from normal traffic. We test each model with the latest emerging attack samples, which have not been learned. The attackers use more diverse variants that destroy some statistical properties, so this method can only correctly classify 16.71% of the samples. The TF-IDF method is slightly improved due to the more fixed syntax of SQL statements, but the attacker can combine different functions or other attacks to achieve their goals. Embedding word vectors can

achieve 76.63% accuracy, which is the most applied approach in recent years. And the method in this paper far exceeds the previous work, which is because synBERT can mine more semantic information. It can distinguish not only between injected statements and plain text but also between normal SQL statements. Our model can correctly classify most of the samples, even new types of attacks that have never been learned.

Table 8. Generalization test results.

Experimental Method	Number of Test Samples	Predicted Number of Correct Samples	Accuracy (%)
Statistical features + MLP_1	796	133	16.71
TF-IDF + CNN_2	796	329	41.33
Word2Vec + CNN_3	796	610	76.63
Method of our paper	796	751	94.35

Note: The subscripts represent the model structure that performs best for a certain vectorization method.

It can be seen that the generalization ability of the algorithm and model proposed in this paper is the best, which further indicates that the model in this paper has the ability to detect unknown attacks.

6. Conclusions and Future Works

We propose a semantic learning and deep learning-based SQL injection attack detection model in this paper and evaluate its basic classification performance. We particularly test the model's generalization performance using a completely new test set. The experimental results show that traditional algorithms based on statistical features and shallow machine learning models perform generally well on various evaluation metrics but fail to detect novel and unknown attacks. The semantic knowledge learning-based word embedding approach is more flexible and can handle this scenario better. The method in this paper has a higher accuracy rate than the other methods, implying that the model has better generalization performance. That is, the model in this paper has a higher detection accuracy for unknown attacks. However, no detection and defense tool (or product) can meet all of your needs. On the one hand, our model is based on detecting network traffic. If the SQL injection attack is the result of user input, it will be visible in HTTP traffic, which synBERT can detect. If a privilege escalation or credential stuffing attack is caused by an insecure database configuration, it should be detected using traffic behavior analysis and mitigated using a combination of other methods. On the other hand, while cookie injection can be detected, attacks against server-side vulnerabilities are helpless. Furthermore, due to the small sample size, the model does not learn enough SQL dialects, which we hope to improve in the future. Another long-term goal is to find ways to reduce model volume while increasing detection speed.

Author Contributions: Conceptualization, D.L. and L.L.; methodology, D.L.; validation, D.L., J.F. and L.L.; formal analysis, J.F.; investigation, D.L.; resources, D.L.; data curation, D.L.; writing—original draft preparation, D.L.; writing—review and editing, J.F. and L.L.; visualization, D.L.; supervision, L.L.; project administration, J.F. and L.L.; funding acquisition, J.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Key Research and Development Project of China (2022-JCJQ-ZD-024-12).

Data Availability Statement: The data used to support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments: The authors are grateful to CVE, CNVD, and Exploit-DB for providing publicly available vulnerability resources.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Chen, D.; Yan, Q.; Wu, C.; Zhao, J. SQL Injection Attack Detection and Prevention Techniques Using Deep Learning. *J. Phys. Conf. Ser.* **2021**, *1757*, 012055. [CrossRef]
2. Salih, N.; Samad, A. Protection Web Applications Using Real-Time Technique to Detect Structured Query Language Injection Attacks. *IJCA* **2016**, *149*, 26–32. [CrossRef]
3. Abdulqadir, H.R.; Zeebaree, S.R.; Shukur, H.M.; Sadeeq, M.M.; Salim, B.W.; Salih, A.A.; Kak, S.F. A Study of Moving from Cloud Computing to Fog Computing. *QAJ* **2021**, *1*, 60–70. [CrossRef]
4. Zhu, Z.; Jia, S.; Li, J.; Qin, S.; Guo, H. SQL Injection Attack Detection Framework Based on HTTP Traffic. In Proceedings of the ACM Turing Award Celebration Conference—China (ACM TURC 2021), Hefei China, 30 July 2021; ACM: Rochester, NY, USA, 2021; pp. 179–185.
5. Jothi, K.R.; Pandey, N.; Beriwal, P.; Amarajan, A. An Efficient SQL Injection Detection System Using Deep Learning. In Proceedings of the 2021 International Conference on Computational Intelligence and Knowledge Economy (ICCIKE), Dubai, United Arab Emirates, 17 March 2021; IEEE: Piscataway, NJ, USA, 2017; pp. 442–445.
6. Li, Q.; Wang, F.; Wang, J.; Li, W. LSTM-Based SQL Injection Detection Method for Intelligent Transportation System. *IEEE Trans. Veh. Technol.* **2019**, *68*, 4182–4191. [CrossRef]
7. Zhuo, Z.; Cai, T.; Zhang, X.; Lv, F. Long Short-term Memory on Abstract Syntax Tree for SQL Injection Detection. *IET Softw.* **2021**, *15*, 188–197. [CrossRef]
8. Tang, P.; Qiu, W.; Huang, Z.; Lian, H.; Liu, G. Detection of SQL Injection Based on Artificial Neural Network. *Knowl.-Based Syst.* **2020**, *190*, 105528. [CrossRef]
9. Li, Q.; Li, W.; Wang, J.; Cheng, M. A SQL Injection Detection Method Based on Adaptive Deep Forest. *IEEE Access* **2019**, *7*, 145385–145394. [CrossRef]
10. Hassan, R.J.; Zeebaree, S.R.M.; Ameen, S.Y.; Kak, S.F.; Sadeeq, M.A.M.; Ageed, Z.S.; AL-Zebari, A.; Salih, A.A. State of Art Survey for IoT Effects on Smart City Technology: Challenges, Opportunities, and Solutions. *AJRCoS* **2021**, *8*, 32–48. [CrossRef]
11. Singh, J.P. Analysis of SQL Injection Detection Techniques. *Theor. Appl. Inf.* **2017**, *28*, 37–55. [CrossRef]
12. Rodríguez, G.E.; Torres, J.G.; Flores, P.; Benavides, D.E. Cross-Site Scripting (XSS) Attacks and Mitigation: A Survey. *Comput. Netw.* **2020**, *166*, 106960. [CrossRef]
13. Fu, X.; Lu, X.; Peltsverger, B.; Chen, S.; Qian, K.; Tao, L. A Static Analysis Framework For Detecting SQL Injection Vulnerabilities. In Proceedings of the 31st Annual International Computer Software and Applications Conference—Vol. 1—(COMPSAC 2007), Beijing, China, 24–27 July 2007; IEEE: Piscataway, NJ, USA, 2007; pp. 87–96.
14. Pan, Y.; Sun, F.; Teng, Z.; White, J.; Schmidt, D.C.; Staples, J.; Krause, L. Detecting Web Attacks with End-to-End Deep Learning. *J. Internet Serv. Appl.* **2019**, *10*, 16. [CrossRef]
15. Shin, Y. Improving the Identification of Actual Input Manipulation Vulnerabilities. In Proceedings of the 14th ACM SIGSOFT Symposium on Foundations of Software Engineering ACM, Portland, OR, USA, 5–11 November 2006.
16. Qu, B.; Liang, B.; Jiang, S.; Ye, C. Design of Automatic Vulnerability Detection System for Web Application Program. In Proceedings of the 2013 IEEE 4th International Conference on Software Engineering and Service Science, Beijing, China, 23–25 May 2013; IEEE: Piscataway, NJ, USA; pp. 89–92.
17. Mui, R.; Frankl, P. Preventing SQL Injection through Automatic Query Sanitization with ASSIST. *Electron. Proc. Theor. Comput. Sci.* **2010**, *35*, 27–38. [CrossRef]
18. Halfond, W.G.J.; Orso, A. AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks. In Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, Long Beach, CA, USA, 7 November 2005; ACM: Rochester, NY, USA, 2005; pp. 174–183.
19. Qing, W.; He, C. The Research of an AOP-Based Approach to the Detection and Defense of SQL Injection Attack. In Proceedings of the 2016 International Conference on Advanced Electronic Science and Technology (AEST 2016), Shenzhen, China, 19–21 August 2016; Atlantis Press: Dordrecht, The Netherlands, 2016.
20. Li, L.; Qi, J.; Liu, N.; Han, L.; Cui, B. Static-Based Test Case Dynamic Generation for SQLIVs Detection. In Proceedings of the 2015 10th International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA), Krakow, Poland, 4–6 November 2015; IEEE: Piscataway, NJ, USA, 2015; pp. 173–177.
21. Katole, R.A.; Sherekar, S.S.; Thakare, V.M. Detection of SQL Injection Attacks by Removing the Parameter Values of SQL Query. In Proceedings of the 2018 2nd International Conference on Inventive Systems and Control (ICISC), Coimbatore, India, 19–20 January 2018; IEEE: Piscataway, NJ, USA, 2018; pp. 736–741.
22. Makiou, A.; Begriche, Y.; Serhrouchni, A. Improving Web Application Firewalls to Detect Advanced SQL Injection Attacks. In Proceedings of the 2014 10th International Conference on Information Assurance and Security, Okinawa, Japan, 28–30 November 2014; IEEE: Piscataway, NJ, USA, 2014; pp. 35–40.
23. Choi, J.; Kim, H.; Choi, C.; Kim, P. Efficient Malicious Code Detection Using N-Gram Analysis and SVM. In Proceedings of the 2011 14th International Conference on Network-Based Information Systems, Tirana, Albania, 7–9 September 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 618–621.
24. Deva Priyaa, B.; Devi, M.I. Fragmented Query Parse Tree Based SQL Injection Detection System for Web Applications. In Proceedings of the 2016 International Conference on Computing Technologies and Intelligent Data Engineering (ICCTIDE'16), Kovilpatti, India, 7–9 January 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–5.

25. Stalmans, E.; Irwin, B. A Framework for DNS Based Detection and Mitigation of Malware Infections on a Network. In Proceedings of the 2011 Information Security for South Africa, Johannesburg, South Africa, 15–17 August 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 1–8.
26. Fang, Y.; Peng, J.; Liu, L.; Huang, C. WOVSQLI: Detection of SQL Injection Behaviors Using Word Vector and LSTM. In Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, Guiyang, China, 16 March 2018; ACM: Rochester, NY, USA; pp. 170–174.
27. Gong, X.; Zhou, Y.; Bi, Y.; He, M.; Sheng, S.; Qiu, H.; He, R.; Lu, J. Estimating Web Attack Detection via Model Uncertainty from Inaccurate Annotation. In Proceedings of the 2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom), Paris, France, 21–23 June 2019; IEEE: Piscataway, NJ, USA, 2019; pp. 53–58.
28. Liu, M.; Li, K.; Chen, T. 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, 18 July 2020; ACM: Rochester, NY, USA; pp. 286–297.
29. Abdulmalik, Y. An Improved SQL Injection Attack Detection Model Using Machine Learning Techniques. *Int. J. Innov. Comput.* **2021**, *11*, 53–57. [CrossRef]
30. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding 2019. In Proceedings of the NAACL-HLT, Minneapolis, MN, USA, 2–7 June 2019.
31. Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.S.; Dean, J. Distributed Representations of Words and Phrases and Their Compositionality. In Proceedings of the NIPS'13: Proceedings of the 26th International Conference on Neural Information Processing Systems, Lake Tahoe, NV, USA, 5–10 December 2013.
32. Linzen, T.; Dupoux, E.; Goldberg, Y. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *TACL* **2016**, *4*, 521–535. [CrossRef]
33. Vig, J. Visualizing Attention in Transformer-Based Language Representation Models 2019. *arXiv* **2019**, arXiv:1904.02679.
34. Hewitt, J.; Manning, C.D. A Structural Probe for Finding Syntax in Word Representations. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Minneapolis, MN, USA, 2–7 June 2019.
35. Coenen, A.; Reif, E.; Yuan, A.; Kim, B.; Pearce, A.; Viégas, F.; Wattenberg, M. Visualizing and Measuring the Geometry of BERT 2019. *arXiv* **2019**. [CrossRef]
36. Farooq, U. Ensemble Machine Learning Approaches for Detection of SQL Injection Attack. *Teh. Glas.* **2021**, *15*, 112–120. [CrossRef]
37. Chen, Z.; Guo, M.; Zhou, L. Research on SQL Injection Detection Technology Based on SVM. *MATEC Web Conf.* **2018**, *173*, 01004. [CrossRef]