People's Democratic Republic of Algeria
Ministery of Higher Education and Scientific Research
Ferhat Abbas University of Setif
Faculty of Sciences
Informatics Department

DISSERTATION

Presented in fulfillment of the requirements of obtaining the degree

Master 2 in Informatics

Specialty: Data Engineering and Web Technologies

THEME

Detecting SQL injections using BERT

Presented by:

ATHMANI RAMI

BOUHEZILA NASSIM

Publicly defended on: dd/mm/yyyy

In front of the jury composed of:

Supervisor: Mr. BENZINE Mehdi. MC UFAS

2022/2023

Dedication

To our parents,

To our grandparents,

To our brothers and sisters,

To our entire family,

To all our friends.

Athmani Rami,

Bouhezila Nassim.

Abstract

Deep learning techniques have improved various domains by using their ability to learn complex patterns from large datasets. In this dissertation, we employed the power of Deep Learning, specifically BERT language model (**Bi**directional **E**ncoder **R**epresentations from **T**ransformers), to resolve the issue of SQL injection attacks on web applications.

The goal of our study is to develop a Deep Learning model using BERT that can accurately identify SQL injections.

Based on the results, our model demonstrated excellent performance; it also indicated that BERT outperforms the compared machine learning models across different evaluation metrics. These results affirm the effectiveness of BERT in detecting SQL injection attacks, underscoring its superior performance in our study.

Résumé

Les techniques d'apprentissage profond ont amélioré divers domaines en utilisant leur capacité à apprendre des complexes patterns à partir de grands ensembles de données. Dans ce mémoire, nous avons utilisé la puissance de l'apprentissage profond, en particulier le modèle de langage BERT (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers), pour résolu le problème des attaques par injection SQL sur les applications web.

L'objectif de notre étude est de développer un modèle d'apprentissage profond en utilisant BERT qui identifier les injections SQL avec précision.

D'après les résultats, notre modèle a démontré d'excellentes performances ; ils ont également indiqués que BERT a surpassé les autres modèles d'apprentissage automatique comparés à travers différents métriques d'évaluation. Ces résultats confirment l'efficacité de BERT dans la détection des attaques par injection SQL, confirmer sa performance supérieure dans notre étude.

ملخص

حسنت تقنيات التعلم العميق مجالات مختلفة باستخدام قدرتها على تعلم الأنماط المعقدة من مجموعات البيانات الكبيرة. في هذه الأطروحة، استخدمنا قوة التعلم العميق، وتحديداً نموذج اللغة "BERT" (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) ، لحل مشكلة هجمات حقن SQL على تطبيقات الويب.

الهدف من دراستنا هو تطوير نموذج التعلم العميق باستخدام BERT الذي يمكنه تحديد هجمات حقن SQL بدقة.

بناءً على النتائج المتحصل عليها، أظهر نموذجنا أداءً ممتازًا؛ كما أشار إلى أن BERT يتفوق في الأداء على نماذج التعلم الآلي التي تم المقارنَة بها عبر مقاييس التقييم المختلفة. تؤكد هذه النتائج فعالية BERT في اكتشاف هجمات حقن SQL ، مما يؤكد أدائها المتفوق في دراستنا.

Table of contents

List of figures

General Introduction

The rapid growth of web applications has revolutionized the way we interact and conduct various activities online. From e-commerce platforms and social networks to financial systems and government portals, web applications have become an integral part of our daily lives. However, with greater dependence on online applications comes an increased danger of cyber attacks with SQL injections being one of the most common and dangerous vulnerabilities.

To mitigate the growing threat of SQL injections, traditional approaches such as input validation and query parameterization have been widely adopted. While these methods provide some level of protection, they often struggle to keep pace with the evolving attack techniques employed by adversaries. Thus, there is a pressing need for more advanced and proactive defense mechanisms to detect and prevent SQL injection attacks.

In recent years, deep learning approaches have emerged as a promising solution in various domains, leveraging their ability to automatically learn complex patterns from large datasets. One such powerful deep learning model is BERT (Bidirectional Encoder Representations from Transformers), originally developed for natural language processing tasks. BERT has proven to be highly effective in capturing the semantic and contextual understanding of text, leading to remarkable performance in tasks such as text classification.

In this research, we propose using the power of BERT-based deep learning models to address the critical issue of SQL injection attacks. Our objective is to develop a reliable and efficient detection model capable of accurately identifying SQL injection attempts in real-time. By using BERT's contextual understanding and semantic representation capabilities, we aim to create a model that can effectively distinguish between normal and SQL malicious queries.

Our thesis is organized as follows:

In Chapter 1, we explore SQL injection attacks, their definitions, types and their detecting techniques, then we head on machine learning and Deep Learning, we present popular

algorithmic approaches in machine learning and explore deep learning architectures in Chapter 2. Chapter 3 is dedicated to the general conception of our work and the materials used, including the dataset and the type of deep learning architecture employed. Furthermore, we cover the preprocessing steps taken to ensure the accuracy and efficiency of our system. In Chapter 4, we discuss the test and evaluation of our model for detecting SQL injection attacks. We use a variety of evaluation metrics, including accuracy, precision, recall, and F1 score. We also compare the performance of our model to other machine learning algorithms and related works.

Chapter 1

SQL Injections

1.1 Introduction

In today's interconnected digital landscape, web applications have become an integral part of our daily lives. They enable us to perform a wide range of tasks, from online shopping and banking to social networking and communication. However, the rise of web applications has also brought about a corresponding increase in security threats and vulnerabilities like gaining unauthorized access, stealing sensitive information, or disrupting services.

The following list shows the Top Five (05) most dangerous web application security risks published by OWASP Top 10 2021 project [1]:

Broken Access Control: refers to inadequate restrictions on what authenticated users can access or perform within an application. It can lead to unauthorized access, privilege escalation, or exposure of sensitive functionalities or data.

Cryptographic Failures: refer to vulnerabilities or weaknesses in the implementation or use of cryptographic techniques and algorithms. These failures can result in the compromise of encrypted data, unauthorized access, or the ability to tamper with cryptographic operations.

Injection Attacks: such as SQL, OS, or LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query, allowing an attacker to execute malicious code or unauthorized actions.

Insecure Design: refers to a fundamental flaw in the architecture or design of a system or application that compromises its security. It involves the failure to incorporate proper security mechanisms, access controls, or threat modeling during the design phase. **Security Misconfiguration:** results from insecure configuration settings, default passwords, open cloud storage, or verbose error messages. Attackers exploit these misconfigurations to gain unauthorized access, extract sensitive information, or launch further attacks.

Focusing on SQL injection attacks, the impact of them on web applications can be severe. As an example, the attackers may have the ability to take control of the entire website or steal sensitive data like usernames, passwords, and credit card numbers. Furthermore, these attacks may result in data loss, website failures, and reputational harm to a company.

1.2 Understanding how web applications work

Web applications are defined as features that are most frequently seen on websites, such as shopping carts, product search and filtering, instant messaging, and social network newsfeeds. They enable users to access sophisticated functionality without having to install or set up software. The web application and users data are stored in a system called DBMS *"Database Management System"*, web apps that use these databases are called Database-driven Web applications.

Database-driven Web applications are very common in today's Web-enabled society. They normally consist of a back-end database with Web pages that contain server-side scripts written in a programming language that is capable of extracting specific information from a database depending on various dynamic interactions with the user. A database-driven Web application commonly has three tiers: Presentation tier (a Web browser or rendering engine, HTML, CSS, JS), logic tier (a programming language, such as C#, PHP, JSP, JS, etc.), storage tier (a database such as Microsoft SQL Server, MySQL, Oracle, etc.). Figure 1.1 Three-tier architecture. illustrates the simple three-tier example.

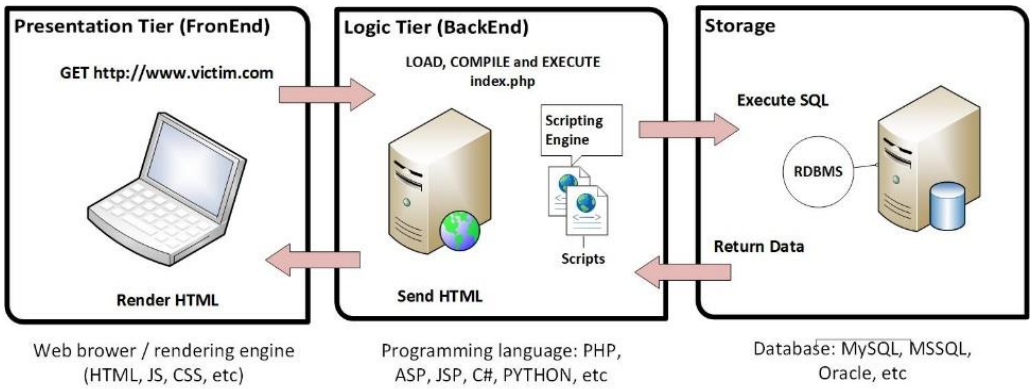

Figure 1.1 Three-tier architecture.

The Web browser (the presentation tier, such as Internet Explorer, Safari, Firefox, etc.) sends requests to the middle tier (the logic tier), which services the requests by making queries and updates against the database (the storage tier).

A fundamental rule in a three-tier architecture is that the presentation tier never communicates directly with the data tier; in a three-tier model, all communication must pass through the middleware tier. Conceptually, the three-tier architecture is linear.

In Figure 1.1, the user fires up his Web browser and connects to http://www.victim.com. The Web server that resides in the logic tier loads the script from the file system and passes it through its scripting engine, where it is parsed and executed. The script opens a connection to the storage tier using a database connector and executes an SQL statement against the database. The database returns the data to the database connector, which is passed to the scripting engine within the logic tier. The logic tier then implements any application or business logic rules before returning a Web page in HTML format to the user's Web browser within the presentation tier. The user's Web browser renders the HTML and presents the user with a graphical representation of the code. All of this happens in a matter of seconds and is transparent to the user.

1.3 How SQL injections work

1.3.1 Definition

SQL injection attack consists of the insertion or "injection" of a SQL query via the input data from the client to the application, they are introduced when software developers create dynamic database queries constructed with string concatenation, which includes user-supplied input. A successful SQL injection exploit can read sensitive data from the database, modify database data (Insert/Update/Delete), execute administration operations on the database (such as shutdown the DBMS), recover the content of a given file present on the DBMS file system and in some cases issue commands to the operating system [2].

Here are some facts regarding damages caused by SQL injection attacks:

1. In 2018, a SQL injection attack on a Canadian cryptocurrency exchange called MapleChange resulted in the loss of almost all of the company's cryptocurrency holdings, which were worth around $6 million.

   MapleChange hack: https://www.ccn.com/maplechange-hacked-all-funds-lost-in-cryptocurrency-exchange-hack/
2. In 2017, a SQL injection attack on Equifax, a major credit reporting agency in the United States, compromised the personal information of over 147 million people.

The breach resulted in a settlement of up to $700 million in damages, including compensation for victims and penalties.

Equifax breach settlement: https://www.npr.org/2019/07/22/744083587/equifax-reaches-700-million-settlement-over-2017-data-breach

3. In 2016, a SQL injection attack on the Philippine Commission on Elections exposed the personal information of over 55 million voters, including their names, addresses, and biometric data. The breach resulted in a class-action lawsuit and cost the commission over $2 million in damages.

   Philippine Commission on Elections breach: https://www.reuters.com/article/us-philippines-election-cyber/philippines-election-commission-hit-by-record-data-breach-idUSKCN0XJ0BQ

In addition to financial damages, SQL injection attacks can also result in reputational damage, loss of customer trust, and legal liabilities. Companies that suffer from SQL injection attacks may face lawsuits, regulatory fines, and a loss of business due to damaged reputation.

Basically, SQL injection process is structured in Four (4) phases:

1. The attacker sends the malicious HTTP request to the web application
2. The malicious code concatenated with the developer's SQL statement
3. The system submits the SQL statement to the backend database.
4. The Database system returns the sensitive data for the Attacker to be exploited after.

As shown in Figure 1.2 describes a login by a malicious user exploiting SQL Injection **vulnerability**. The Administrator will be authenticated on the application after typing: employee id=112 and password=admin. The attacker is trying to find a SQL injection vulnerability to log in to the application by adding a tautology to the developer's SQL statement. [3]

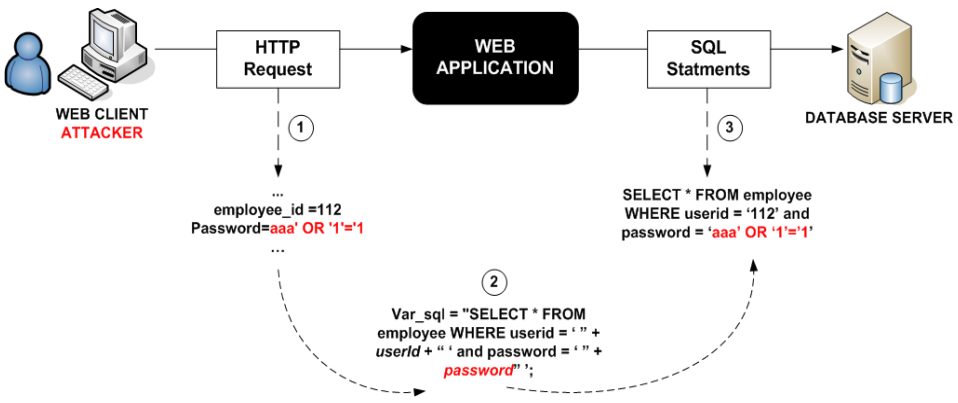

Figure 1.2 Example of a SQL injection attack.

The above SQL statement is always true because of the Boolean tautology that the attacker appended (OR 1=1) so, he will access the web application as an administrator without knowing the right password.

1.4 Techniques of SQL injections

1.4.1 Tautologies

This type of attack injects SQL tokens into the conditional query statement to be evaluated as always true. This type of attack is used to bypass authentication control and access to data by exploiting vulnerable input fields which use WHERE clause For example [5], consider the following SQL query used for user authentication:

SELECT * FROM users WHERE username = 'admin' AND password = 'password'

An attacker could inject a tautology into the password field, such as **' OR '1'='1'**

This would cause the query to become:

SELECT * FROM users WHERE username = 'admin' AND password = '' OR '1'='1'

Since the condition '1'='1' is always true, the query would return all users in the database, including the admin account. The attacker could then log in as the admin without knowing the correct password.

1.4.2 Error-based SQL injection

In error-based SQL injection, attackers can use SQL queries that trigger errors in order to learn about the structure and contents of a database. For example, an attacker could send an SQL query to the application that intentionally causes an error, and the error message returned by the database will contain information about the structure of the database or the content of the queried table.

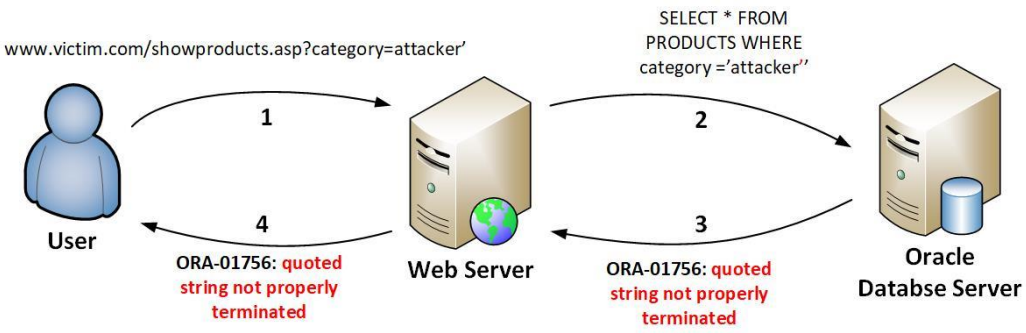

Figure 1.3 How Information flows during an SQL injection error.

According to Figure 1.3, the following occurs during a SQL injection error:

1. The user sends a request in an attempt to identify a SQL injection vulnerability. In this case, the user sends a value with a single quote appended to it.
2. The Web server retrieves user data and sends a SQL query to the database server. In this example, we can see that the SQL statement created by the Web server includes the user input and forms a syntactically incorrect query due to the two terminating quotes.
3. The database server receives the malformed SQL query and returns an error to the Web server.
4. The Web server receives the error from the database and sends an HTML response to the user. In this case, it sent the error message, but it is entirely up to the application how it presents any errors in the contents of the HTML response [4].

The following error is usually an indication of a MySQL injection vulnerability:

Warning: mysql_fetch_array(): supplied argument is not a valid MySQL result resource in /var/www/victim.com/showproduct.php on line 8

The preceding example illustrates the scenario of a request from the user which triggers an error in the database. Depending on how the application is coded, the response returned in step 4 will be constructed and handled as a result of one of the following:

- The SQL error is displayed on the page and is visible to the user from the Web browser.
- The SQL error is hidden in the source of the Web page for debugging purposes.
- Redirection to another page is used when an error is detected.
- An HTTP error code 500 (Internal Server Error) or HTTP redirection code 302 is returned.
- The application handles the error properly and simply shows no results, perhaps displaying a generic error page.

1.4.2.1 Mysql database errors

MySQL can be executed in many architectures and Operating systems. An Apache Web server running PHP on a Linux operating system forms a common configuration, but we can find it in many other scenarios as well.

In the example of my sql error shown in figured in Figure 1.3, the attacker injected a single quote in a GET parameter and the PHP page sent the SQL statement to the database. The following Figure 1.4 shows a fragment of PHP code that displays the vulnerability [4]:

```
<?php

//Connect to the database

//Error checking in case the database is not accessible

mysql_connect( "[database]", "[user]", "[password]") or die("Could
not connect:". mysql_error() );

//Select the database

mysql_select_db ("[database_name]");

//We retrieve category value from the GET request

$category = $_GET["category"];

//Create and execute the SQL statement

$result = mysql_query("SELECT * from products where
category='$category'");

//Loop on the results

while ($row = mysql_fetch_array($result, MYSQL_NUM)) {printf("ID: %s
Name: %s", $row[0], $row[1]);}

//Free result set

mysql_free_result($result);

?>
```

Figure 1.4 SQL injection vulnerability in PHP code.

The code shows that the value retrieved from the GET variable is used in the SQL statement without sanitization. If an attacker injects a value with a single quote, the resultant SQL statement will be:

SELECT * FROM products WHERE category='attacker''

The preceding SQL statement will fail and the mysql_query function will not return any value. Therefore, the *$result* variable will not be a valid MySQL result resource. In the following line of code, the *mysql_fetch_array($result, MYSQL_NUM)* function will fail and

PHP will show the warning message that indicates to an attacker that the SQL statement could not be executed.

In the preceding example, the application does not disclose details regarding the SQL error, and therefore the attacker will need to devote more effort in determining the correct way to exploit the vulnerability.

PHP has a built-in function called *mysql_error*, which provides information about the errors returned from the MySQL database during the execution of a SQL statement. In Figure 1.5, the following PHP code displays errors caused during the execution of the SQL query:

```
<?php
//Connect to the database
//Error checking in case the database is not accessible
mysql_connect("[database]", "[user]", "[password]") or die("Could not connect:". mysql_error());
//Select the database
mysql_select_db("[database_name]");
//We retrieve category value from the GET request
$category = $_GET["category"];
//Create and execute the SQL statement
$result = mysql_query("SELECT * from products where category='$category'");
if (!$result) { //If there is any error
//Error checking and display
die('<p>Error:'. mysql_error(). '</p>');
} else {
// Loop on the results
while ($row = mysql_fetch_array($result, MYSQL_NUM)) {printf("ID: %s Name:%s", $row[0], $row[1]);
}//Free result set
mysql_free_result($result);}
?>
```

Figure 1.5 handle query error with mysql_error() function in PHP.

When an application runs the preceding, the code that catches database errors and the SQL query fails, the returned HTML document will include the error returned by the database. If an attacker modifies a string, parameter by adding a single quote the server will return output similar to the following:

Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ’’’at line 1

The preceding output provides information regarding why the SQL query failed. If the injectable parameter is not a string and therefore is not enclosed between single quotes, the resultant output would be similar to this:

Error: Unknown column ‘attacker’ in ‘where clause’

The behavior in MySQL server is identical to Microsoft SQL Server; because the value is not enclosed between quotes, MySQL treats it as a column name. The SQL statement executed was along these lines:

SELECT * FROM products WHERE idproduct=attacker

MySQL cannot find a column name called attacker and therefore returns an error.

This is the code snippet from the PHP script shown earlier in charge of error handling:

```
//If there is any error
//Error checking and display
    if (!$result) {
    die('<p>Error:'. mysql_error(). '</p>');}
```

In this example, the error is caught and then displayed using the *die()* function. The PHP *die()* function prints a message and gracefully exits the current script. Other options are available for the programmer, such as redirecting to another page:

```
//If there is any error
//Error checking and redirection
if (!$result) {
    header("Location:http://www.victim.com/error.php");}
```

1.4.3 Blind SQL Injection

Blind SQL injection is a technique of SQL Injection attack that asks the database true or false questions and determines the answer based on the application's response. This attack is often used when the web application is configured to show generic error messages but has not mitigated the code that is vulnerable to SQL injection.

When an attacker exploits SQL injection, sometimes the web application displays error messages from the database complaining that the SQL Query's syntax is incorrect. Blind SQL injection is nearly identical to normal SQL Injection, the only difference being the way the data is retrieved from the database. When the database does not output data to the web page, an attacker is forced to steal data by asking the database a series of true or false questions. This makes exploiting the SQL Injection vulnerability more difficult, but not impossible [5].

An attacker may verify whether a sent request returned true or false in a few ways:

1.4.3.1 Content-based

Using a simple page, which displays an article with a given ID as the parameter, the attacker may perform a couple of simple tests to determine if the page is vulnerable to SQL Injection attacks.

Example URL: http://newspaper.com/items.php?id=2, this URL access sends the following query to the database:

```
SELECT title, description, body FROM items WHERE ID = 2
```

The attacker may then try to inject a query that returns *'false'*:

```
http://newspaper.com/items.php?id=2 and 1=2
```

Now the SQL query should look like this:

```
SELECT title, description, body FROM items WHERE ID = 2 and 1=2
```

If the web application is vulnerable to SQL Injection, then it probably will not return anything. To make sure, the attacker will inject a query that will return 'true':

```
http://newspaper.com/items.php?id=2 and 1=1
```

If the content of the page that returns 'true' is different than that of the page that returns 'false', then the attacker is able to distinguish when the executed query returns true or false.

Once this has been verified, the only limitations are privileges set up by the database administrator, different SQL syntax, and the attacker's imagination.

1.4.3.2 Time-based

This type of blind SQL injection relies on the database pausing for a specified amount of time, then returning the results, indicating successful SQL query execution. Using this method, an attacker enumerates each letter of the desired piece of data using the following logic:

- *If the first letter of the first database's name is an 'A', wait for 10 seconds.*
- *If the first letter of the first database's name is a 'B', wait for 20 seconds. etc.*

Using some time-taking operation e.g. *BENCHMARK()*, will delay server responses if the expression is True.

BENCHMARK(5000000,ENCODE('MSG','by 5 seconds')) will execute the ENCODE function 5000000 times.

Depending on the database server's performance and load, it should take just a moment to finish this operation. The important thing is, from the attacker's point of view, to specify a high-enough number of BENCHMARK() function repetitions to affect the database response time in a noticeable way.

Here is an example combination of both queries:

```
1 UNION SELECT IF(SUBSTRING(user_password,1,1) =
CHAR(50),BENCHMARK(5000000,ENCODE('MSG','by 5 seconds')),null) FROM users
WHERE user_id = 1;
```

If the database response took a long time, we may expect that the first user password character with *user_id = 1* is character '2', *(CHAR(50) == '2')*

Using this method for the rest of the characters, it's possible to enumerate entire passwords stored in the database. This method works even when the attacker injects the SQL queries and the content of the vulnerable page doesn't change.

Obviously, in this example, the names of the tables and the number of columns were specified. However, it is possible to guess them or check with a trial and error method.

Conducting Blind SQL Injection attacks manually is very time-consuming, but there are a lot of tools that automate this process. One of them is SQLMap partly developed within the

OWASP grant program. On the other hand, tools of this kind are very sensitive to even small deviations from the rule. This includes:

- Scanning other website clusters, where clocks are not ideally synchronized,
- WWW services where the argument acquiring method was changed, e.g. from /index.php?ID=10 to /ID,10

If the attacker is able to determine when their query returns True or False, then they may fingerprint the RDBMS. This will make the whole attack much easier. If the time-based approach is used, this helps determine what type of database is in use. Another popular method to do this is to call functions, which will return the current date. MySQL, MSSQL, and Oracle have different functions for that, respectively *now()*, *getdate()*, and *sysdate()* [5].

1.4.4 Union-based SQL injections

The UNION operator is used in SQL to combine the results of two or more SELECT statements into a single result set. When a web application contains a SQL injection vulnerability that occurs in a SELECT statement, the attacker can often employ the UNION operator to perform a second, entirely separate query, and combine its results with those of the first.

All major DBMS products support UNION. It is the quickest way to retrieve arbitrary information from the database in situations where query results are returned directly [6]. Example of an application that enabled users to search for books based on author, title, publisher, and other criteria. Searching for books published by Wiley causes the application to perform the following query:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley'
```

Suppose that this query returns the following set of results:

AUTHOR	TITLE	YEAR
Litchfield	The Database Hacker's Handbook	2005
Anley	The Shellcoder's Handbook	2007

A far more interesting attack would be to use the *UNION* operator to inject a second SELECT query and append its results to those of the first. This second query can extract data from a different database table.

For example, entering the search term:

Wiley' UNION SELECT username,password,uid FROM users-- causes the application to perform the following query:

SELECT author,title,year FROM books WHERE publisher = 'Wiley' UNION SELECT username ,password, uid FROM users--'

This returns the results of the original search followed by the contents of the users table:

AUTHOR	TITLE	YEAR
Litchfield	The Database Hacker's Handbook	2005
Anley	The Shellcoder's Handbook	2007
admin	r00tr0x	0
cliff	Reboot	1

This simple example demonstrates the potentially huge power of the UNION operator when employed in a SQL injection attack. However, before it can be exploited in this way, two important provisos need to be considered [6]:

- When the results of two queries are combined using the UNION operator, the two result sets must have the same structure. In other words, they must contain the same number of columns, which have the same or compatible data types, appearing in the same order.
- To inject a second query that will return interesting results, the attacker needs to know the name of the database table that he wants to target, and the names of its relevant columns.

1.5 SQL Injection defense techniques

With user input channels being the main vector for such attacks, the best approach is controlling and vetting user input to watch for attack patterns by applying the following main prevention methods [7]:

1.5.1 Escaping

The developer must use always character-escaping functions for user-supplied input provided by each database management system (DBMS). This is done to make sure the DBMS never confuses it with the SQL statement provided by the developer.

For example, use the mysql_real_escape_string() in PHP to avoid characters that could lead to an unintended SQL command. A modified version for the login bypass scenario would look like the following in Figure 1.6:

```
   $db_connection = mysqli_connect("localhost", "user", "password", "db");

   $username          =          mysqli_real_escape_string($db_connection,
$_POST['username']);

   $password          =          mysqli_real_escape_string($db_connection,
$_POST['password']);

   $query = "SELECT * FROM users WHERE username = '" . $username. "' AND
password = '" . $password . "'";
```

Figure 1.6 Character-escaping in PHP code example.

Previously, the code would be vulnerable to adding an escape character (\) in front of the single quotes. However, having this small alteration will protect against an illegitimate user and mitigate SQL injection.

1.5.2 Input validation

The validation process is aimed at verifying whether or not the type of input submitted by a user is allowed. Input validation makes sure it is the accepted type, length, format, and so on. Only the value that passes the validation can be processed. It helps counteract any commands inserted in the input string. In a way, it is similar to looking to see who is knocking before opening the door.

Validation should not only be applied to fields that allow users to type in input, meaning developers should also take care of the following situations in equal measure:

- Use regular expressions as whitelists for structured data (such as name, age, income, survey response, zip code) to ensure strong input validation.
- In case of a fixed set of values (such as drop-down list, radio button), determine which value is returned. The input data should match one of the offered options exactly.

The below shows how to carry out table name validation.

```
switch ($tableName) {

    case 'fooTable': return true;

  case 'barTable': return true;

    default: return new BadMessageException('unexpected value provided as table name');

}
```

The *$tableName* variable can then be directly appended—it is now widely known to be one of the legal and expected values for a table name.

In the case of a drop-down list, it's very easy to validate the data. Assuming the developers want a user to choose a rating from 1 to 5, change the PHP code to something like this:

```
<?php
if(isset($_POST["selRating"])) {
    $number = $_POST["selRating"];
      if( (is_numeric($number)) && ($number > 0) && ($number < 6) ){
          echo "Selected rating: " . $number;
      } else { echo "The rating has to be a number between 1 and 5!"; } ?>
```

The developers have added two simple checks:

1. It has to be a number (the is_numeric() function).
2. He requires that $number to be bigger than 0 and smaller than 6, which leaves him with a range of 1–5.

Data that is received from external parties has to be validated. This rule applies not only to the input provided by Internet users but also to suppliers, partners, vendors, or regulators. These vendors could be under attack and send malformed data even without their knowledge.

1.5.3 Parameterized queries

Parameterized queries are a means of pre-compiling an SQL statement so that you can then supply the parameters in order for the statement to be executed. This method makes it possible for the database to recognize the code and distinguish it from input data.

The user input is automatically quoted and the supplied input will not cause a change of the intent, so this coding style helps mitigate an SQL injection attack.

PHP 5.1 up versions present a better approach when working with databases: PHP Data Objects (PDO). PDO adopts methods that simplify the use of parameterized queries. Additionally, it makes the code easier to read and more portable since it operates on several databases, not just MySQL.

The code in Figure 1.7 uses PDO with parameterized queries to prevent the SQL injection vulnerability:

```
<?php

$id = $_GET['id'];

$db_connection = new PDO('mysql:host=localhost;dbname=sql_injection_example', 'dbuser', 'dbpasswd');

//preparing the query

$sql = "SELECT username FROM users WHERE id = :id";

$query = $db_connection->prepare($sql);

$query->bindParam(':id', $id);

$query->execute();

//getting the result

$query->setFetchMode(PDO::FETCH_ASSOC);

$result = $query->fetchColumn();

print(htmlentities($result)); ?>
```

Figure 1.7 Parameterized queries using PDO in PHP code example.

1.5.4 Web application firewalls WAF

One of the best practices to identify SQL injection attacks is having a web application firewall (WAF). A WAF operating in front of the web servers monitors the traffic which goes in and out of the web servers and identifies patterns that constitute a threat. Essentially, it is a barrier put between the web application and the Internet.

A WAF operates via defined customizable web security rules. These sets of policies inform the WAF what weaknesses and traffic behavior it should search for. So, based on that information, a WAF will keep monitoring the applications and the GET and POST requests it receives to find and block malicious traffic [7].

WAFs provide efficient protection from a number of malicious security attacks such as:

- SQL injection
- Cross-site scripting (XSS)
- Session hijacking
- Distributed denial of service (DDoS) attacks
- Cookie poisoning
- Parameter tampering

1.5.5 Detection using machine learning

Artificial **I**ntelligence can enhance the speed and efficiency of SQL injection detection. By automating the process of query analysis and classification, AI algorithms can rapidly scan vast amounts of incoming queries, flagging potential threats in real-time.

One of the many machine-learning algorithms that can be used for SQL injection detection as an example is SVM (Support Vector Machines).

SVM is a supervised machine-learning algorithm that is commonly used for binary classification tasks; it aims to find an optimal hyperplane that separates the legitimate queries from the malicious ones by maximizing the margin between them.

To train a model using SVM, a systematic process that include **Dataset Preparation**, **Feature Extraction**, **Dataset Splitting**, **and Model Training** must be followed. During the training point, the SVM model learns the decision boundary based on the provided features and their corresponding labels.

1.6 Conclusion

As web applications become increasingly complex, SQL injection attacks remain a persistent and evolving threat. However, various techniques and tools have been developed to detect and prevent SQL injection attacks like input validation and sanitization, parameterized queries, and security-focused coding practices.

Unfortunately, because of the large variation in the pattern of SQL injection attacks, it is often unable to protect databases. Therefore, it is recommended, to apply the above-mentioned techniques in combination with Machine Learning and AI tools.

Chapter 2

Deep Learning

2.1 Introduction

Artificial intelligence (AI) is a field of computer science that aims to create intelligent systems capable of performing tasks that typically require human intelligence. From self-driving cars to voice assistants, AI has made remarkable strides, transforming the way we live, work, and interact with technology. AI comprises two fundamental branches: machine learning (ML) and deep learning (DL).

In this chapter, we will explore common methods of machine learning, including supervised, unsupervised, and reinforcement learning. We will also discuss popular algorithmic approaches in machine learning, moving to explore deep learning and its architectures. Our focus will be on understanding the fundamental concepts behind these methods and algorithms.

2.2 Machine learning

Machine Learning is an integral part of artificial intelligence that allows computers to learn and adapt without having to be explicitly programmed. It involves developing computer algorithms capable of automatically adjusting and improving their performance in response to data.

At its core, Machine Learning is a data analysis technique that automates the creation of analytical models. Through algorithms, computers can uncover hidden insights and patterns, without being explicitly instructed where to look. This iterative nature allows models to independently adapt and make reliable and repeatable decisions by learning from previous computations.

Although Machine Learning algorithms have been around for some time, recent advancements in technology have significantly accelerated their application. The ability to rapidly process vast and complex volumes of data, coupled with affordable computational power and data storage, has unlocked new possibilities. As a result, Machine Learning is now widely used across diverse domains, including the development of autonomous vehicles, online recommendation systems, customer sentiment analysis, and fraud detection.

The renewed interest in Machine Learning is driven by the increasing availability of diverse datasets and the decreasing costs associated with computational resources. This enables the generation of models that can quickly analyze large volumes of data, providing businesses with valuable insights and opportunities to identify lucrative prospects or mitigate potential risks.

2.2.1 Types of machine learning

Machine learning can be categorized into three main types: supervised learning, unsupervised learning, and reinforcement learning.

2.2.1.1 Supervised learning

Supervised learning involves training machine learning models on labeled data. The model learns to make predictions or classify new data by finding patterns and relationships between the input features and the desired outputs. Examples of supervised learning algorithms include linear regression, decision trees, and support vector machines (SVMs) [1].

Supervised learning addresses two types of problems: classification problems and regression problems.

Classification is a type of supervised learning problem where the goal is to predict a discrete class label for each data point. For example, the goal might be to predict whether an image contains a cat or a dog, or whether a patient has cancer or not. Classification problems can be solved using a variety of machine learning algorithms, including decision trees and support vector machines.

Regression is a type of supervised learning problem where the goal is to predict a continuous value for each data point. For example, the goal might be to predict the price of a house or the height of a person. Regression problems can be solved using a variety of machine learning algorithms, including linear regression and polynomial regression.

2.2.1.2 Unsupervised learning

Unsupervised learning involves training models on unlabeled data. The algorithms learn to identify patterns, structures, and relationships in the data without explicit guidance. Common tasks in unsupervised learning include clustering, where similar data points are grouped together, and dimensionality reduction, which aims to reduce the complexity of data while preserving important information. Examples of unsupervised learning algorithms include k-means clustering, hierarchical clustering, and principal component analysis (PCA) [9].

2.2.1.3 Reinforcement

Reinforcement learning involves training agents to interact with an environment and learn optimal behaviors through trial and error. The agent receives feedback in the form of rewards or penalties based on its actions, and its objective is to maximize cumulative rewards over time. Reinforcement learning is commonly used in scenarios where an agent needs to make sequential decisions, such as in game playing, robotics, and autonomous vehicle control. Popular reinforcement learning algorithms include Q-learning, policy gradients, and deep Q-networks (DQNs).

It's worth noting that these types of machine learning are not mutually exclusive, and hybrid approaches that combine multiple types can be used to tackle complex problems. Additionally, there are other specialized areas within machine learning, such as semi-supervised learning and transfer learning, which further expand the capabilities of machine learning algorithms [10].

2.2.2 Machine learning algorithms

Machine learning algorithms can be thought of as powerful tools that allow us to unlock the potential of data. By applying these algorithms to various problems, we can uncover hidden patterns, detect anomalies, classify data into different categories, and even make accurate predictions about future outcomes.

2.2.2.1 Linear regression

Linear regression is a machine learning algorithm that uses a line to predict a numerical value based on input variables. As shown in Figure 2.1 the line is found by minimizing the difference between the predicted and actual values. Linear regression is simple to understand and interpret, and it is commonly used in a variety of domains [8].

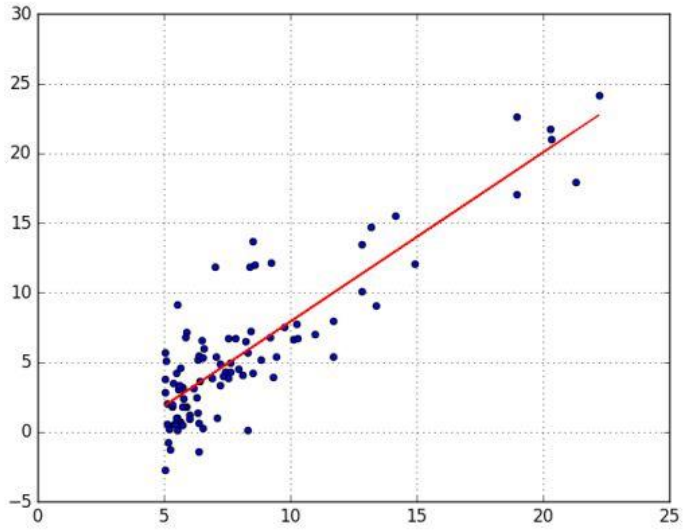

Figure 2.1 Graphical representation of linear regression.

2.2.2.2 Logistic Regression

Logistic regression is a machine-learning algorithm that predicts the probability of an instance belonging to a specific class. It is a supervised learning method that uses the logistic function to model the relationship between the input variables and the probability of the binary outcome as shown in Figure 2.2. Logistic regression is commonly used in a variety of domains, including customer behavior analysis, fraud detection, and medical diagnosis. Its advantages lie in its simplicity, interpretability, and ability to handle linear and nonlinear relationships between variables [8].

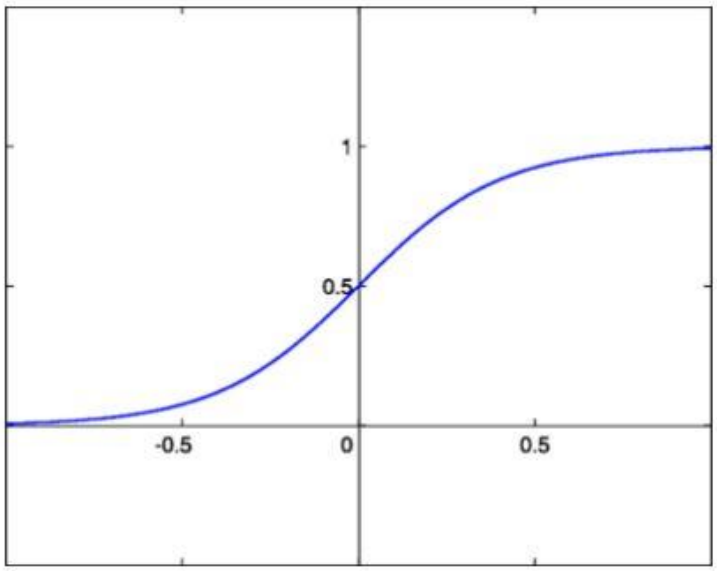

Figure 2.2 Graphical representation of logistic regression.

2.2.2.3 Support vector machines

Support vector machines (SVMs) are a machine learning algorithm that can be used for both classification and regression tasks. As shown in Figure 2.3 SVMs work by finding the optimal hyperplane that separates data points of different classes with the maximum margin. This means that the hyperplane is as far away as possible from any data points on either side. SVMs can handle both linearly separable and non-linearly separable data by using kernel functions to implicitly map the data into a higher-dimensional space.

Their key strengths lie in their ability to handle high-dimensional data and flexibility in choosing different kernel functions [8].

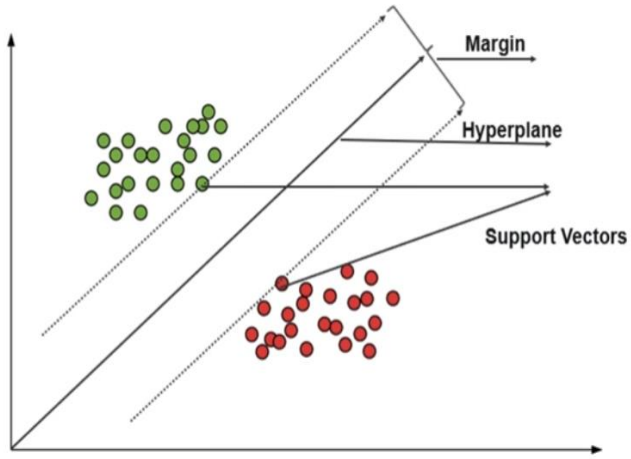

Figure 2.3 Graphical representation of Support Vector Machines.

2.2.2.4 K-Means

K-means is an unsupervised machine learning algorithm that groups data points into clusters based on their similarities. The algorithm works by iteratively assigning data points to the cluster with the nearest centroid, and then updating the centroids based on the mean of the assigned points as shown in Figure 2.4. The number of clusters, K, is predefined by the user. K-means is a simple and computationally efficient algorithm, making it widely used for clustering tasks.

K-means finds applications in various domains, including customer segmentation, image compression, and document clustering.

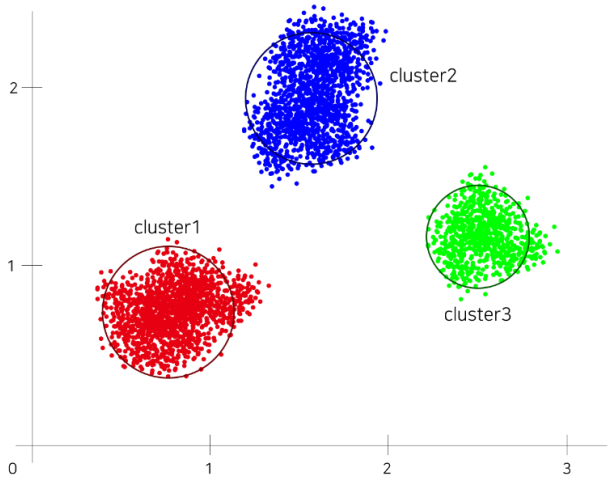

Figure 2.4 Graphical representation of k means.

2.2.3 Machine learning applications

Machine learning is a rapidly evolving field with a wide range of applications. It has been used to improve healthcare, transportation, finance, retail, energy, and agriculture.

Healthcare and Medicine: In healthcare, machine learning is used to diagnose diseases, recommend treatments, and personalize care.

Automotive Industry: In transportation, machine learning is used to develop autonomous vehicles and improve traffic management.

Financial Services: In finance, machine learning is used to detect fraud, assess risk, and make investment decisions.

Energy and Utilities: In energy, machine learning is used to optimize energy consumption, predict outages, and improve grid reliability.

Agriculture: In agriculture, machine learning is used to predict crop yields, detect pests, and optimize irrigation.

These are just some of the industries that machine learning had been applied to, machine learning is a powerful tool that has the potential to transform many industries.

2.3 Deep learning

Deep learning (DL) is a subset of machine learning (ML) that uses multilayer artificial neural networks to model and solve complex problems. These networks are designed to simulate the

behavior of neurons in the human brain, enabling them to process and learn from large amounts of unstructured data.

Deep learning algorithms automatically learn to recognize patterns and features in data by analyzing and adjusting the weights and biases of network interconnected nodes. This process, called training, involves optimizing network parameters to minimize errors and improve accuracy.

Deep learning has grown in popularity in recent years due to its ability to process large and diverse datasets, achieve cutting-edge performance in many fields, and achieve breakthroughs in areas such as computer vision, natural language processing, and speech recognition [11].

2.3.1 Activation functions

Activation functions are an essential component of neural networks as they introduce non-linearity, allowing the network to learn and approximate complex patterns in the data. Here are some commonly used activation functions:

Sigmoid: The sigmoid function is a smooth S-shaped curve that maps the input to a value between 0 and 1. It is often used in binary classification problems or as an output activation function in models that require probability-like outputs.

ReLU (Rectified Linear Unit): ReLU is a piecewise linear function that returns the input as if it is positive, and 0 otherwise. It is widely used in hidden layers of deep neural networks due to its simplicity and computational efficiency.

Tanh (Hyperbolic Tangent): The hyperbolic tangent function is similar to the sigmoid function but maps the input to a value between -1 and 1. It provides stronger non-linearity than the sigmoid function and is often used in recurrent neural networks (RNNs) and convolutional neural networks (CNNs).

Softmax: Softmax is a specialized activation function used in multi-class classification problems. It takes a vector of real numbers as input and normalizes them into a probability distribution over multiple classes, where the sum of the probabilities is 1.

These activation functions serve different purposes and may be more suitable for specific tasks or network architectures. Choosing the appropriate activation function depends on the nature of the problem and the desired behavior of the neural network.

2.3.2 Deep Neural Networks

Neural networks are inspired by the neurons in the human brain. They are designed to discover complex patterns in data. Neural networks consist of many interconnected nodes, which are organized into layers. As described in Figure 2.5 The input layer receives data, the hidden layers process the data, and the output layer produces a prediction. Neural networks can be used for a variety of tasks, such as classification and regression [12].

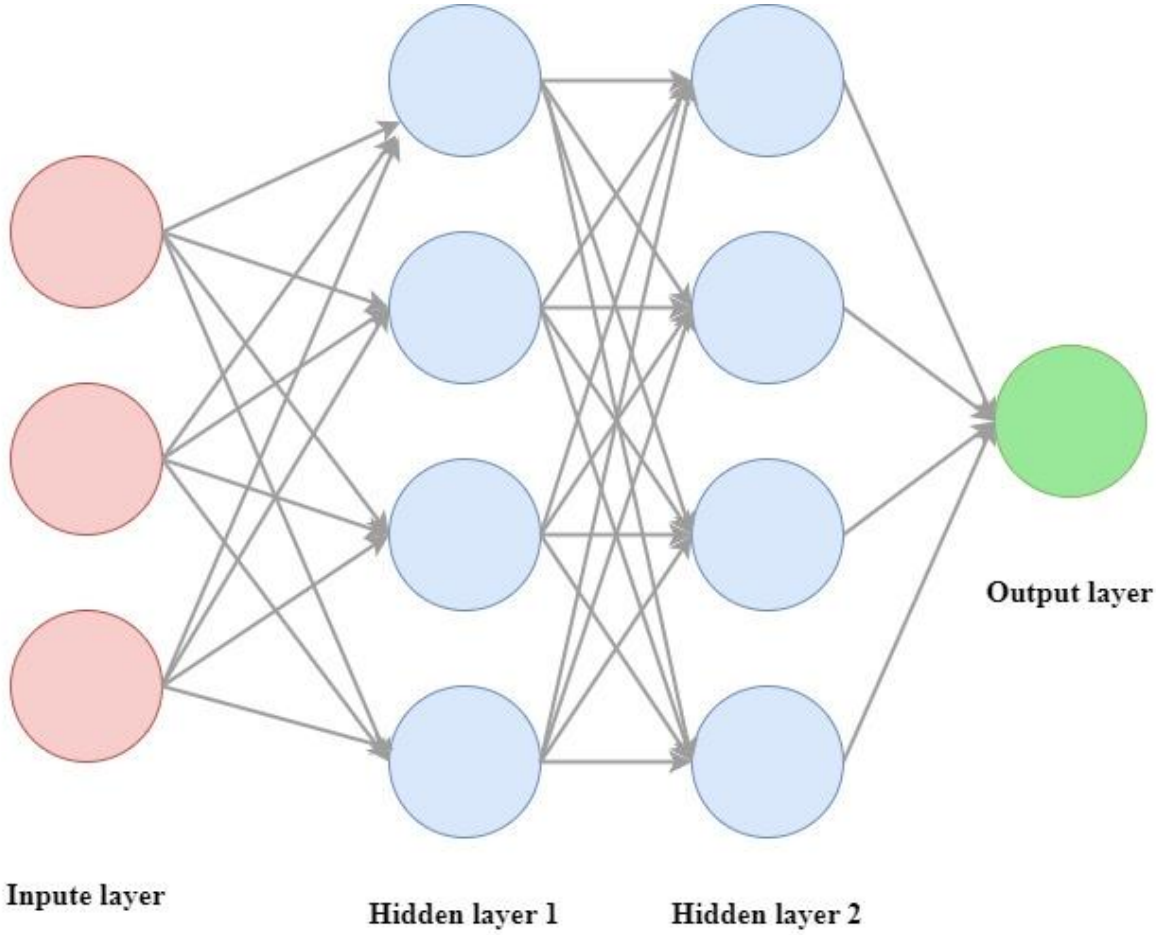

Figure 2.5 Schematic representation of a neural network.

2.3.3 Deep learning architectures

Deep Learning is a growing field with applications that span across several use cases. In each use case, a different architecture is predominant and gives the best efficiency.

There are many different types of deep learning architectures, many of which are derived from original architectures. Some of the most popular ones are Convolutional Neural Networks

(CNNs), Recurrent Neural Networks (RNNs), and Long Short-Term Memory Networks (LSTMs).

For the purposes of this discussion, we will talk about Recurrent Neural Networks, Long Short-Term Memory Networks, Gated Recurrent Units (GRU) and focus on one type of architecture known as transformers, which have gained popularity in recent years for their ability to process sequential data with parallelization and attention mechanisms.

2.3.3.1 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of neural network that is particularly effective in processing sequential data. Unlike traditional neural networks, which process input data independently, RNNs have a feedback mechanism that allows them to retain and utilize information from previous steps in the sequence. This makes them well-suited for tasks such as natural language processing, speech recognition and machine translation [11].

RNNs work by processing each step in the sequence one at a time. At each step, the RNN takes as input the current input data and the output from the previous step. The RNN then uses this information to calculate a new output. This output is then used as input for the next step, and so on [13].

Some popular variants of RNNs include LSTMs (Long Short-Term Memory) and GRUs (Gated Recurrent Units). LSTMs and GRUs are designed to address the vanishing gradient problem that can occur in traditional RNNs, which hinders their ability to capture long-term dependencies. LSTMs incorporate memory cells and gating mechanisms that enable them to selectively retain and update information over time. GRUs have gating mechanisms that control the flow of information within the network, allowing them to capture long-term dependencies efficiently [14].

2.3.3.2 Long Short-Term Memory Networks

Traditional RNNs suffer from a problem known as the vanishing gradient problem, which can hinder their performance when dealing with long-term dependencies.

The vanishing gradient problem arises during the training of RNNs when the gradients used for learning become extremely small as they propagate backward through time. This occurs because the gradient calculation involves multiplying a series of weight matrices, and the gradient values tend to diminish exponentially with each multiplication. As a result, the network

struggles to capture and propagate information from earlier time steps, limiting its ability to model long-term dependencies in the data [15].

To address the vanishing gradient problem, the Long Short-Term Memory (LSTM) architecture was introduced. LSTM networks incorporate specialized memory cells that allow for better information retention and controlled information flow.

The key innovation of LSTM is the inclusion of memory cells, which serve as a way to store and access information over long time scales. These memory cells are equipped with three main components: an input gate, a forget gate, and an output gate. These gates regulate the flow of information into and out of the memory cells, allowing them to selectively retain and forget information as needed.

The input gate determines how much new information is stored in the memory cells, while the forget gate controls which information is discarded. The output gate determines the information to be passed on to the next time step. By carefully controlling the flow of information, LSTM networks can effectively address the vanishing gradient problem and capture long-term dependencies in sequential data [16].

The LSTM architecture has been widely successful in various applications, including natural language processing, speech recognition, machine translation, and video analysis. Its ability to model long-term dependencies makes it a crucial component in many state-of-the-art deep learning models.

2.3.3.3 Gated Recurrent Units

The Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) architecture that addresses the vanishing gradient problem of traditional RNNs. GRUs were introduced in 2014 by Cho et al. to capture long-term dependencies in sequential data while mitigating the issue of information loss over time.

GRUs consist of two main gates: the reset gate and the update gate. The reset gate determines how much of the previous hidden state should be forgotten, while the update gate determines how much of the new candidate state should be incorporated into the current hidden state. These gates allow GRUs to selectively retain and update information over time, facilitating the learning of complex dependencies in sequential data.

GRUs have several advantages over traditional RNNs and even other gated architectures like LSTMs. They have fewer parameters, making them computationally more efficient and easier to train. Additionally, GRUs have been found to perform competitively or even outperform LSTMs on certain tasks, while requiring less training time and data [17].

Overall, GRUs are a powerful and versatile RNN architecture that offers several advantages over traditional RNNs and LSTMs.

2.3.3.4 Transformers

As we already mentioned that RNNs have several limitations, including difficulty in parallelization and difficulty in capturing long-term dependencies. These limitations have led to the development of alternative models, such as transformers.

The Transformer is a neural network architecture that was proposed by Vaswani et al. in 2017. It is a powerful model that has revolutionized the field of natural language processing (NLP). Transformer models introduced a new approach to sequence modeling without recurrent connections, which makes them more efficient and easier to train than traditional RNN-based models. Transformer models have been widely adopted and have become the state-of-the-art models in various NLP tasks, including machine translation, question answering, and text generation [18].

The Transformer architecture adopts an encoder-decoder structure. As shown in Figure 2.6, the encoder maps an input sequence of symbol representations (x1, ..., xn) to a sequence of continuous representations (z1, ..., zn). The decoder, using the encoded representation z, generates an output sequence (y1, ..., ym) by producing one symbol at a time. This process is auto-regressive, as the model incorporates previously generated symbols as additional input for generating the next symbol. The Transformer architecture follows this overall design, employing stacked self-attention and point-wise, fully connected layers for both the encoder and decoder [18].

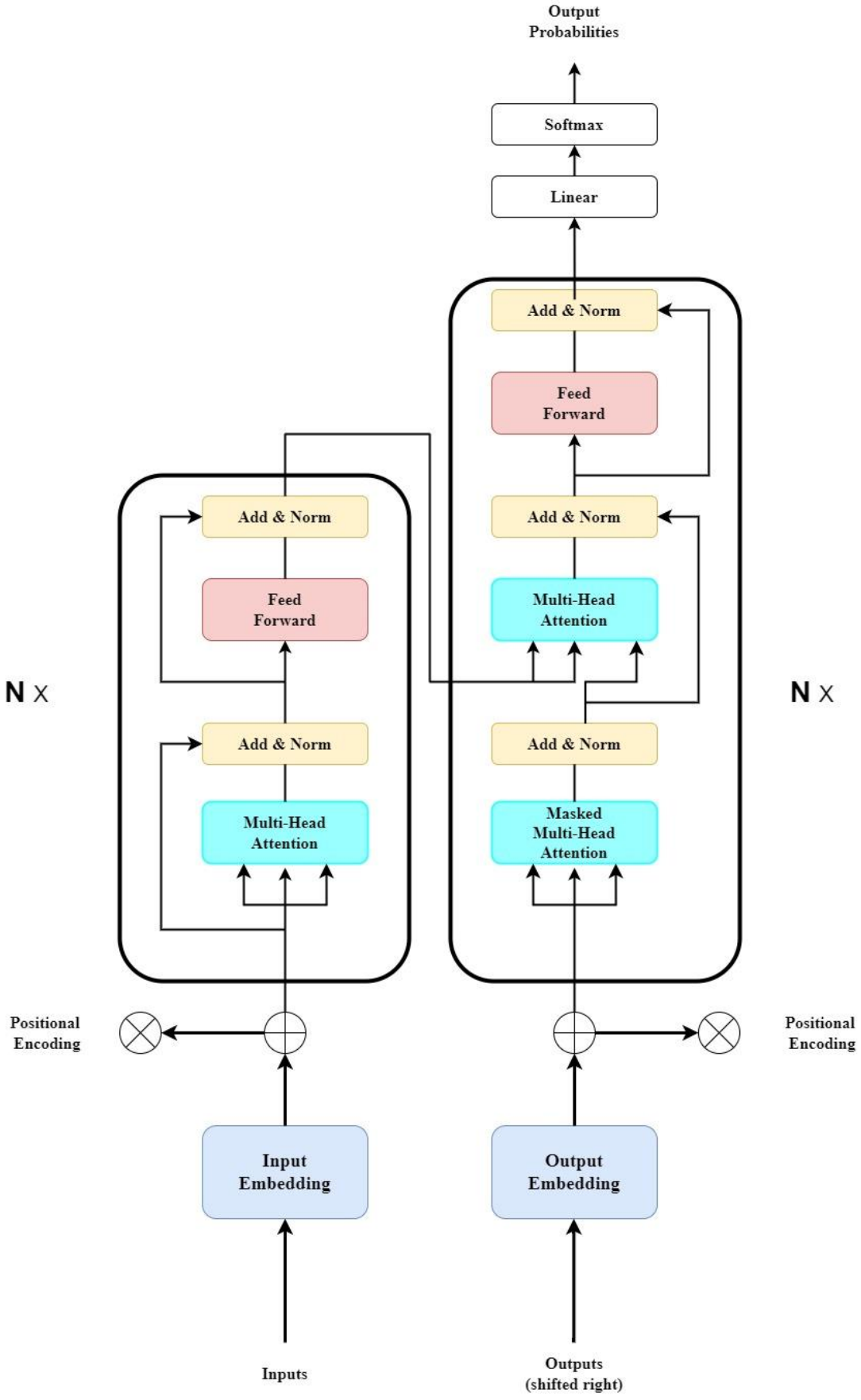

Figure 2.6 Architecture of transformers.

Encoder

The encoder is made up of a stack of N = 6 identical layers. Each layer has two sub-layers. The first sub-layer is a multi-head self-attention mechanism, and the second is a simple feed-forward network. Each of these two layers is followed by a normalization layer. To facilitate the connections in the model, all sub-layers produce outputs of dimention dmodel = 512.

Decoder

The decoder is also made up of a stack of N = 6 identical layers. Each layer has three sub-layers. In addition to the two sub-layers in each encoder the decoder inserts a third sub-layer which is a multi-head attention over the output of the encoder. Similar to the encoder each of the three sub-layers is followed by a normalization layer. In the decoder stack, the self-attention sub-layer is modified to make sure positions don't pay attention to the positions that come after them, using a technique called masking. This, along with the fact that the output embeddings are shifted by one position, ensures that the predictions for a particular position only rely on the already known outputs at positions before it.

Attention

An attention function can be defined as a mapping between a query and a set of key-value pairs, producing an output. In this mapping, the query, keys, values, and output are all vectors. The output is computed by calculating a weighted sum of the values, where the weight assigned to each value is determined by a compatibility function between the query and the corresponding key.

Scaled Dot-Product Attention

To compute the scaled dot-product attention, the query and key vectors are multiplied together to obtain a similarity score for each pair. The dot product represents how well the query aligns with each key. These scores are then scaled by the square root of the dimension of the key vectors to avoid overly large values.

Next, the scaled scores are passed through a softmax function, which normalizes the scores and assigns a weight to each value vector. The softmax function ensures that the weights sum up to 1, making the attention mechanism a proper distribution.

Finally, the values are multiplied by their corresponding weights and summed up to produce the output of the attention mechanism. This output captures the relevant information from the value vectors based on the similarity between the query and key vectors.

Multi-Head Attention

Instead of performing a single attention function with keys, values, and queries of dmodel dimensions, it has been found beneficial to linearly project the queries, keys, and values h times. Each projection uses different learned linear transformations to convert them into dk, dk, and dv dimensions, respectively. Subsequently, the attention function is applied in parallel to these projected versions of queries, keys, and values, resulting in dv-dimensional output values. These values are then concatenated and projected again, producing the final values.

Feed-Forward Networks

In addition to the attention sub-layers, each layer in the encoder and decoder includes a fully connected feed-forward network. This network operates on each position independently and uniformly. It comprises two linear transformations with a ReLU activation function in between. [12]

Embeddings and Softmax

Similar to other sequence transduction models, learned embeddings are used to convert input tokens and output tokens into dmodel-dimensional vectors. The decoder output is transformed into predicted probabilities for the next token using a learned linear transformation and softmax function.

Positional Encoding

Positional encoding adds position-related information to the input data, allowing the model to understand the order of tokens. It involves incorporating fixed-length vectors into the input embeddings to capture sequential dependencies. This enables transformers to effectively process sequential data for tasks like machine translation and language understanding.

2.3.4 Deep learning applications

Deep learning (DL) is a powerful tool that has revolutionized many fields by delivering unprecedented accuracy and efficiency in processing complex data. One of the main strengths

of DL is its ability to automatically extract relevant features and patterns from large and diverse datasets without the need for manual feature extraction. This makes deep learning particularly suitable for applications such as image and video processing, natural language understanding, speech recognition, and autonomous decision-making.

For example, DL has made significant progress in image and video recognition tasks such as object recognition, face recognition, and scene understanding. Convolutional Neural Networks (CNNs), a popular DL architecture for image and video analytics, can learn hierarchical representations of image features and achieve cutting-edge performance on many benchmarks. Likewise, in natural language processing, DL has achieved breakthroughs in tasks such as sentiment analysis, machine translation, and question answering. Recurrent Neural Networks (RNNs) and Transformers are common DL architectures for processing sequential and textual data.

Deep learning is also being applied in many other domains such as Healthcare, Finance and Transportation. For example, DL models have been developed for medical image analysis, drug discovery, and personalized treatment planning. In finance, deep learning algorithms are used for fraud detection, credit risk assessment, and trading strategy optimization. In transportation, deep learning has been used for autonomous driving, traffic prediction, and route planning. These are just a few examples of the wide range of applications DL can enable, and the field is constantly evolving with new breakthroughs and innovations.

2.4 Conclusion

In conclusion, this chapter has provided an overview of both deep learning and machine learning. We have discussed their fundamental concepts, algorithms, and architectures, shedding light on these powerful techniques in the field of artificial intelligence.

Chapter 3

Conception and Implementation

3.1 Introduction

In this chapter, we will discuss the design and implementation of our model for detecting SQL injections using deep learning. We will describe the general conception of our work and the materials used, including the dataset and the type of deep learning architecture employed. Furthermore, we will cover the preprocessing steps taken to ensure the accuracy and efficiency of our system. By detailing our approach to design and implementation, we aim to provide a comprehensive understanding of our methodology for detecting SQL injections through the use of deep learning.

3.2 General conception of the solution

The deep learning model that we have developed will be used for detecting SQL injection attacks in web applications. The model is based on the Bidirectional Encoder Representations from Transformers (BERT) architecture, which has been fine-tuned on a dataset of SQL injection attacks and normal SQL queries. The model is designed to function as a middleware layer (API or a WAF) between the web application and the database server as shown in Figure 3.1 Sql injection Detection Tool conception and architecture.

Our **Sqli Detection Model** analyzes incoming queries and detects any suspicious patterns that may indicate an SQL injection attack, it can be used on Web Application Firewalls (WAF) or as an API. Once the SQL injection is detected, the tool can either block the request or alert the system administrator, depending on the configuration.

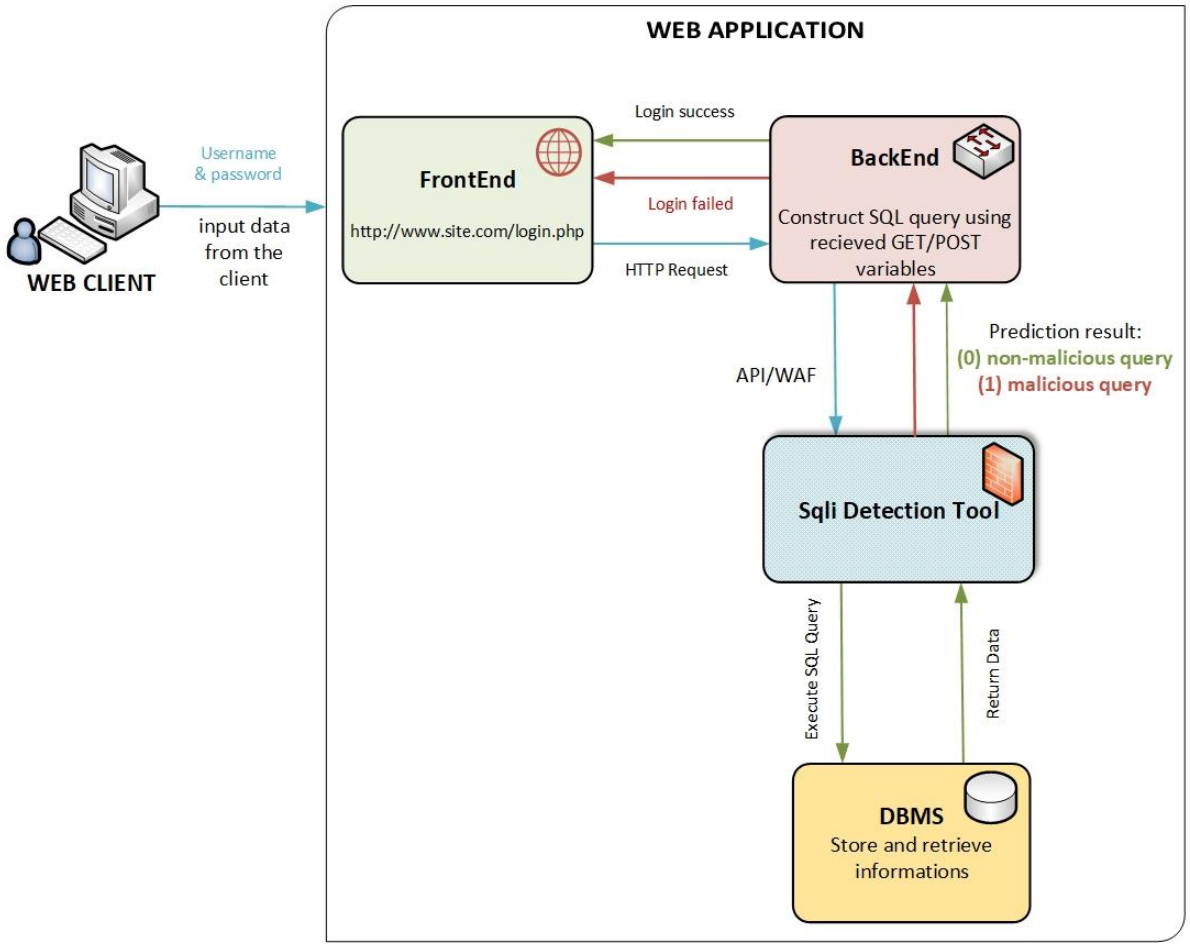

Figure 3.1 Sql injection Detection Tool conception and architecture.

3.3 Chosen model: BERT

BERT (Bidirectional Encoder Representations from Transformers) is a state-of-the-art language model that has revolutionized natural language processing tasks. It is based on transformer architecture, which enables it to effectively capture contextual information from input text. Unlike traditional models that process text sequentially, BERT takes into account the entire context surrounding each word by using a bidirectional approach. By pre-training on large amounts of unlabeled text data, BERT learns to generate high-quality word representations that encode rich semantic and syntactic information. These pre-trained representations can then be fine-tuned on specific tasks, such as detecting SQL injections in our case. With its ability to understand the nuanced context of language, BERT has demonstrated

exceptional performance on various natural language processing tasks, making it a suitable choice for enhancing the detection and prevention of SQL injection attacks in this study.

BERT has two variants: BERT-base and BERT-large, which differ in the number of layers and parameters. BERT-base has 12 transformer layers and 110 million parameters, while BERT-large has 24 transformer layers and 340 million parameters [19].

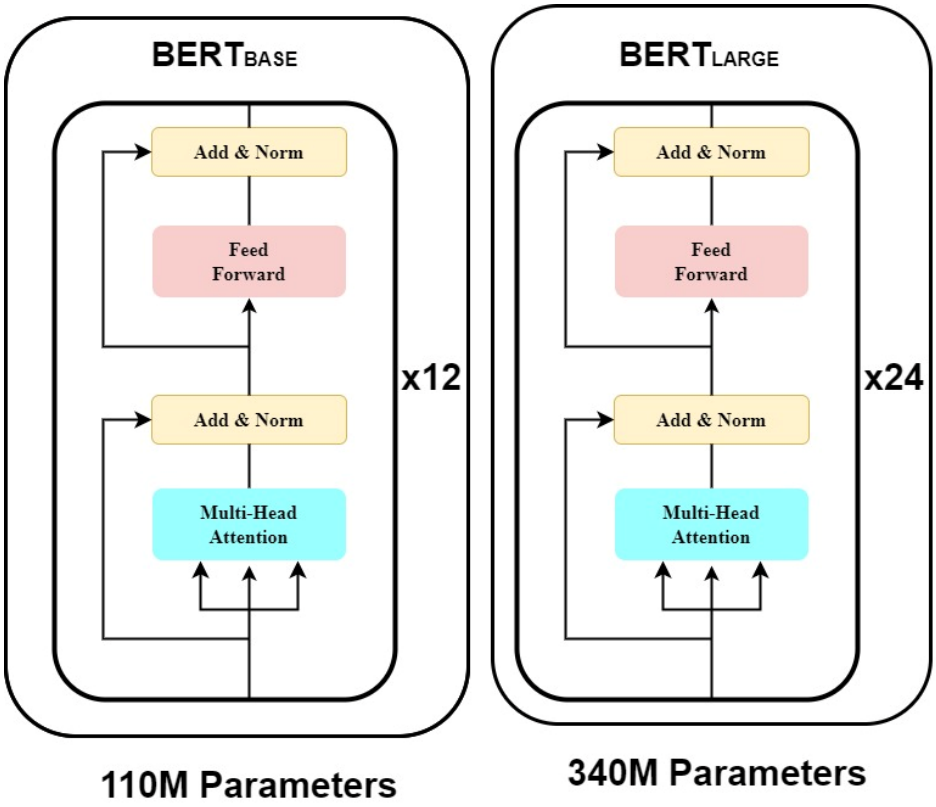

Figure 3.2 BERT model size.

In this study, the choice of the BERT-Base model was driven by the size of our dataset, which consisted of approximately 22,000 samples. Given the limited size of the dataset, using BERT-Large would not have been the most suitable option.

3.3.1 BERT architecture

BERT uses the Transformer architecture, which is an attention mechanism designed to learn contextual relationships between words or sub-words in a text. In its original form, the Transformer consists of two mechanisms: an encoder that processes the text input and a decoder

that generates predictions for the task. However, since BERT's objective is to generate a language model, only the encoder mechanism is necessary.

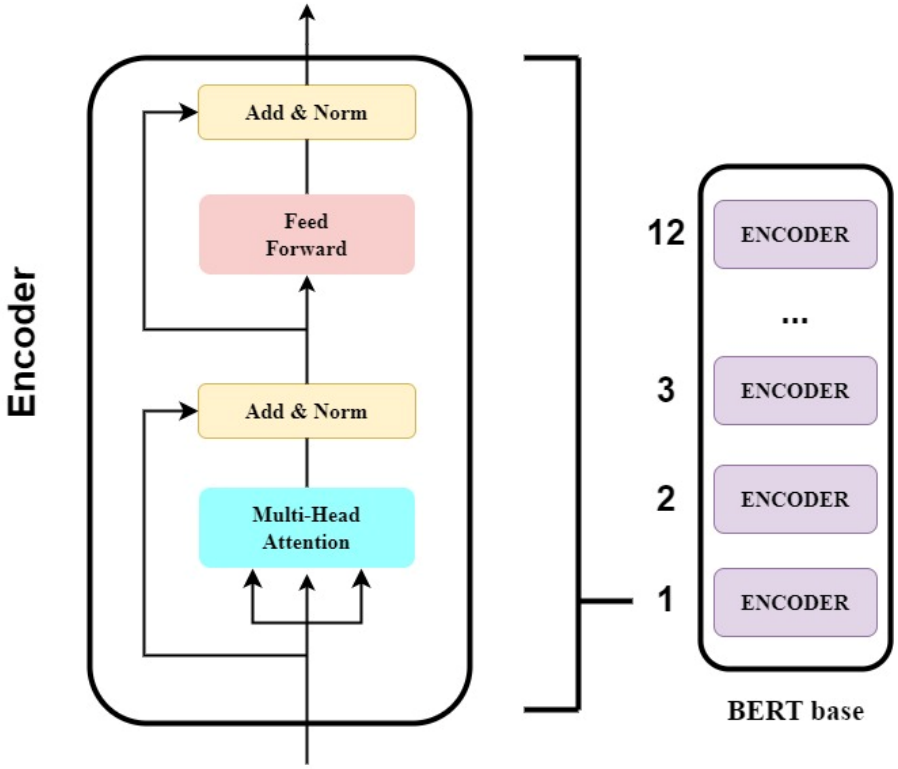

Figure 3.3 BERT model architecture.

The encoder in BERT comprises a stack of N = 6 identical layers, with each layer consisting of two sub-layers. The first sub-layer is a multi-head self-attention mechanism, which allows the model to attend to different positions within the input sequence simultaneously. This mechanism employs multiple attention heads to capture dependencies between words or sub-words in a text. The second sub-layer is a simple feed-forward network, which processes the outputs from the previous self-attention layer. This network consists of fully connected layers and applies a linear transformation to each position independently. Both sub-layers in each encoder layer are followed by a normalization layer. This normalization helps in stabilizing the learning process by normalizing the outputs of each sub-layer. Additionally, to ensure smooth connectivity between layers in the model, all sub-layers produce outputs of dimension dmodel = 512, ensuring consistent input and output dimensions throughout the architecture.

3.3.2 BERT for text classification

When processing input, BERT expects a sequence of tokens with a maximum length of 512 tokens. The sequence can be divided into one or two segments. The first token of the sequence is a special token that holds a special classification embedding. For text classification tasks, BERT considers the final hidden state h of the first token as the representation of the entire sequence. This hidden state captures the information from the entire input sequence and serves as a condensed representation. To make predictions, a simple softmax classifier is added on top of BERT. The classifier employs a task-specific parameter matrix W to compute the probability of each label c given the representation h.

3.3.3 Why BERT was chosen

In the context of SQL injection detection, BERT was chosen for its ability to learn rich representations of text data, including queries, comments, and other textual elements that are typically associated with SQL injection attacks. BERT has been shown to outperform other state-of-the-art models on various NLP tasks [20], making it a promising candidate for SQL injection detection. Its transformer-based architecture allows it to process input sequences in a bidirectional manner and generate contextualized word representations. Fine-tuning BERT on a labeled dataset of SQL queries allowed us to develop a model that can detect SQL injection attacks with high accuracy.

3.3.4 Fine-tuning BERT for SQL injection detection:

To use BERT for SQL injection detection, we fine-tuned the pre-trained BERT model on a labeled dataset of normal SQL queries and SQL injections. During fine-tuning, the model was trained to predict whether a given query is a SQL injection or not. The fine-tuning process involved adjusting the weights of the classification layer while keeping the weights of the pre-trained BERT layers fixed.

3.4 Presentation of development tools

3.4.1 Programming language

3.4.1.1 Python 

Python is a popular programming language in the field of machine learning and artificial intelligence due to its simple syntax, extensive libraries, and ease of use. Python provides a variety of libraries for machine learning such as TensorFlow, PyTorch, and Keras, making it a

go-to language for many data scientists and machine learning practitioners. Its libraries provide an extensive range of functionalities, from data preprocessing to complex neural network architectures [21].

Furthermore, Python's community is continuously contributing to its open-source libraries, ensuring a broad range of features and capabilities. Python is also known for its versatility as it can be used not only for machine learning but also for web development, scientific computing, and data analysis. However, it is important to note that while Python is a popular choice, it is not the only programming language used in machine learning. Other languages, such as R and Java, are also used for machine learning tasks [22].

3.4.2 Development environment

3.4.2.1 Google Colab Pro

Since training deep learning models often requires high-performance hardware or can be time-consuming, we selected Google Colab Pro as the platform for our project. Google Colab Pro is a cloud-based service that provides access to a powerful GPU and RAM, which allows us to train our model in a reasonable amount of time.

Specifically, Google Colab Pro provides access to a Tesla K80 GPU, which has 12 GB of GDDR5 VRAM and 4992 CUDA cores. This GPU is suitable for training deep learning models with moderate to high computational requirements.

In addition, **Google Colab Pro** also provides access to TPUs, which are specifically designed for accelerating deep learning computations. TPUs are available for users on a case-by-case basis and require a separate application process.

The TPU offered by Google Colab Pro is the TPU v3-8, which has 8 TPU cores and 64 GB of High Bandwidth Memory (HBM). This TPU is designed for high-throughput deep learning workloads and can accelerate training times by orders of magnitude compared to a traditional GPU.

Overall, the availability of both GPU and TPU on Google Colab Pro made it a suitable platform for our deep learning project, enabling us to train and test our model efficiently and effectively."

Additionally, Google Colab Pro provides a user-friendly interface that allows us to write and execute our code using Jupyter notebooks. This platform also offers other useful features such as version control, collaboration tools, and cloud storage for our data and code.

3.4.2.2 **Jupyter notebook**

Jupyter notebook is an open-source web application that allows users to create and share documents that contain live code, equations, visualizations, and narrative text. It supports over 40 programming languages, including Python, R, and Julia, making it a versatile tool for data analysis and machine learning [23].

Jupyter notebooks are interactive and allow users to execute code in a step-by-step manner, making it easy to debug and analyze results. They also support the use of Markdown, a markup language for text formatting, making it easy to create readable and well-structured documents. Jupyter notebooks are widely used in the data science community due to their flexibility, interactivity, and ease of use [24].

3.5 Dataset

In order to train a successful and effective deep learning model, the dataset must be carefully processed, to achieve this, we needed to find a Dataset consisting of samples divided into two classes: "malicious queries" and "non-malicious queries". By using this Dataset, the model can learn to distinguish between the two classes and accurately identify SQL injection attacks.

During our search for a suitable Datasets, we came across the "SQL Injection Dataset" list on the Kaggle platform [25]. We found a list of datasets that were created by “Syed Hussain” that contain Three (03) versions:

- ✓ **SQLi.csv** (723.15 kB) contains **3951** samples with **78%** classified as normal queries and **28%** as malicious queries.
- ✓ **SQLiV2.csv** (3.61 MB) contains **33726** samples with **66%** classified as normal queries and **34%** as malicious queries.
- ✓ **SQLiV3.csv** (2.32 MB) contains **30873** samples with **62%** classified as normal queries and **37%** as malicious queries and **1%** as other.

At first vision we choose the **SQLiV2** Dataset because of the size of samples in it in comparison to others, we trained our model with it in the first time but unfortunately, the results

were not satisfied while predicting normal queries. After analyzing the situation, we found that there were no normal queries in the dataset, only free text in place flagged as normal queries.

Because of the big issue in **SQLiV2.csv**, we tried the **SQLiV3.csv** Dataset**.** After reviewing it, we identified certain deficiencies that need to be cleared using some preprocessing steps like:

- ✓ Remove any unnecessary Strings or characters in the SQL statements. We found two commas („) at the end of all the queries, we removed these unnecessary commas using a Register-Expression technique on a python script.
- ✓ Remove not valid empty columns. We need only two valid columns, the **Statement** and the **Label**, in that Dataset we found two empty and not valid columns that were removed using the Office Excel Software.
- ✓ Remove empty and free text Rows. We found many empty and free text rows that were removed using the Office Excel Software.
- ✓ Remove wrong SQL queries. We found not valid SQL statements that were identified and cleared manually.

By applying the above-preprocessed steps, we finally came up with a partitioned version that has **11308 (≈ 50%)** "non-malicious queries" against **11291 (≈ 50%)** "malicious queries".

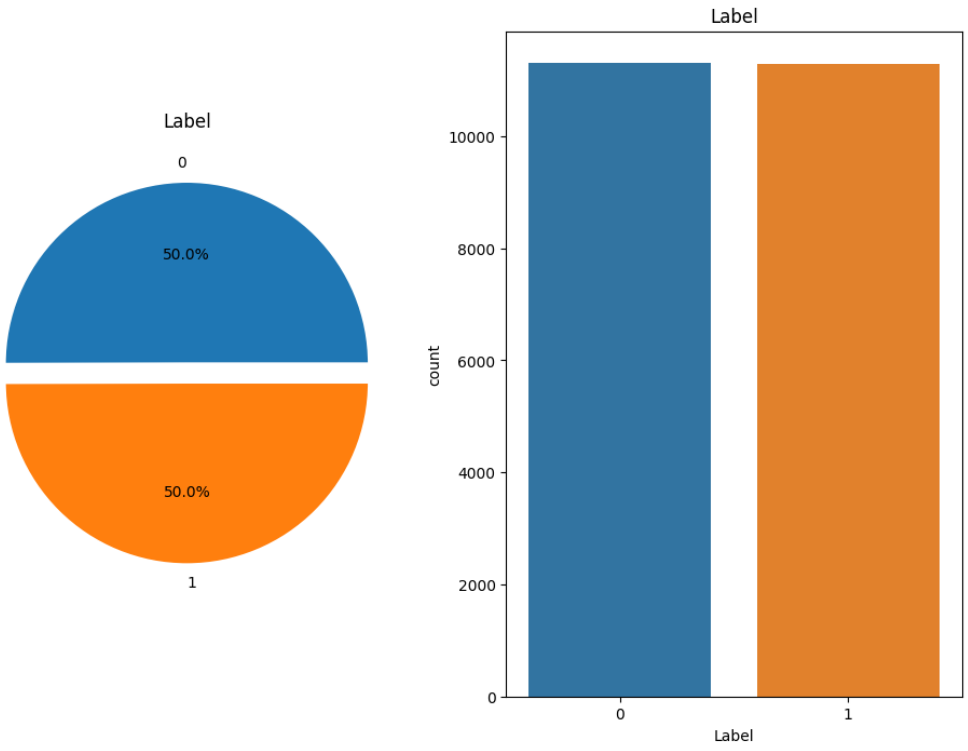

Figure 3.4 Dataset query classes distribution.

After clearing the different deficiencies, we came up with a new valid preprocessed Dataset that has sufficient diversity in both categories and is ready to be used in the next step. We ensured that our deep learning model would only train on preprocessed samples that belong to one of the two classes and could, therefore, learn to differentiate between malicious and non-malicious queries effectively.

3.6 Code and implementation

The implementation of a deep learning model can be a challenging task, especially when it involves complex architectures and large datasets. In this section, we will present the code and implementation details of our SQL injection detection model based on the BERT architecture.

3.6.1 Import necessary libraries and tools

Importing necessary libraries and tools is an essential step when working on any data science project. These libraries provide functionality for common data manipulation, exploration, and deep learning tasks. In this project, we used a number of libraries to preprocess and classify text data as the following:

numpy: NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices. We used this library to work with numerical data in our project.

ktrain: ktrain is a lightweight wrapper for the Keras deep learning library to help simplify the training of neural networks. We used this library to build and train our text classification model.

pandas: Pandas is a library used for data manipulation and analysis. It provides data structures for efficiently storing and manipulating large datasets. We used this library to read in and preprocess our text data.

chardet: Chardet is a Python library used for character encoding detection. We used this library to ensure that our text data is properly encoded before processing.

matplotlib: Matplotlib is a data visualization library used for creating static, animated, and interactive visualizations in Python. We used this library to create visualizations of our data and model performance.

seaborn: Seaborn is a data visualization library based on matplotlib. It provides a high-level interface for creating informative and attractive statistical graphics. We used this library to visualize the distribution of our data.

pickle: Pickle is a Python module used for serializing and de-serializing Python objects. We used this library to save our trained model for future use.

3.6.2 Split and preprocess data for the BERT model

```
sentences = df['Sentence'].tolist()

labels = df['Label'].tolist()

(x_train, y_train), (x_test, y_test), preproc =
text.texts_from_array( sentences,labels,
preprocess_mode='bert', maxlen=500,
val_pct=0.2,class_names=list(set(labels))
)
```

Figure 3.5 Split data into training and testing sets and preprocess data for BERT model

In Figure 3.5 Split data into training and testing sets and preprocess data for BERT model, the code is splitting the data into training and testing sets and prepares it to be used for training the model. It starts by extracting the 'Sentence' and 'Label' columns from the pandas dataframe and creating two lists out of them. Next, the 'texts_from_array' method from the ktrain library is used to convert the data into arrays that can be used for training a BERT model. The 'preprocess_mode' parameter is set to 'bert', which means that the data will be preprocessed according to the requirements of the BERT model. val_pct=0.2 indicates that 20% of the data will be used for validation during training. The 'maxlen' parameter is set to 500, which means that the maximum length of a sentence is set to 500 tokens. The 'class_names' parameter is set to the unique labels in the training set, which will be used to create a mapping between the label values and their corresponding names. The method returns four variables: x_train and x_test

are the preprocessed arrays of sentences, y_train and y_test are the label arrays, and preproc is a preprocessor object that was used to preprocess the data.

3.6.3 Build the BERT model

```
  model = text.text_classifier('bert', (x_train, y_train),
preproc=preproc)
```

Figure 3.6 Build BERT model Python code.

In this code line, a BERT (Bidirectional Encoder Representations from Transformers) model is being built using the text_classifier function from the ktrain library. This function takes the name of the model as the first argument (in this case, 'bert'), the training data (x_train, y_train), and the preprocessing object preproc as input.

3.6.4 Train the BERT model

```
  learner = ktrain.get_learner(model=model,
                               train_data=(x_train, y_train),
                               val_data=(x_test, y_test),
                               batch_size=6)
  learner.fit onecycle(lr=2e-5, epochs=4)
```

Figure 3.7 Train BERT model Python code.

In the "Train BERT model" section, the model is trained using ktrain's get_learner method and the one-cycle policy, which involves training the model with a learning rate that linearly increases for the first half of the epochs and then linearly decreases for the second half of the epochs.

First, get_learner is called, which creates a Learner object for the specified model and training data (x_train, y_train). It also includes validation data (x_test, y_test) to monitor the model's performance during training, and the batch_size is set to 6.

Then, the fit_onecycle method is called on the learner object. This method trains the model using the one-cycle policy for a specified lr (learning rate) and number of epochs. In this case, the learning rate is set to 2e-5 and the number of epochs is 4.

During training, the model's loss and accuracy are displayed for each epoch. The goal is to minimize the loss and maximize the accuracy on the validation set to create a well-performing model.

3.6.5 Make predictions with the BERT model

```
predictor = ktrain.get_predictor(learner.model, preproc)

# make predictions

samples = [

"1'; DROP TABLE users;--;",
"INSERT INTO users (username, password) VALUES ('testuser',
'testpassword'); DROP TABLE users;",
"SELECT COUNT(*) FROM users WHERE username = 'admin' OR 1 = 1",
"UPDATE users SET password = 'newpassword' WHERE username =
'admin';" ,
"select * from generate_series ( 5980,5980,case
when ( 5980 = 5063 ) then 1 else 0 end ) limit 1--",
"SELECT TOP 3 * FROM growth SELECT * FROM catch 3SELECT * FROM
mainly",
"SELECT * FROM users WHERE username = '' OR 1:1",
"INSERT INTO column ( white, does, certain, curious, first, our
) VALUES ( 'rose', 'anyone'. close', 'remove', 'force', 'feet',
'fell' )",
"SELECT * FROM users WHERE username = '' OR 1=2 --' AND password =
'input_password'"]

prediction = predictor.predict(samples)
```

Figure 3.8 Make predictions with the trained model.

This code block shows how to use the BERT model trained to detect SQL injection attacks to make predictions on new data. The get_predictor() function loads the trained BERT model and pre-processing pipeline, which are necessary to make predictions on new data. The predict() function takes a single input and returns the predicted label.

3.7 Choice of hyperparameters

The choice of hyperparameters plays a critical role in the design and performance of our system for detecting SQL injections using deep learning. In this section, we discuss the key hyperparameters we selected and the reasoning behind these choices.

3.7.1 Preprocessing hyperparameters

Max Length: We set the maximum length of input sequences to 500. This value was determined based on the analysis of the dataset, ensuring that most SQL injection statements can be adequately captured within this limit.

Preprocess Mode: We utilized the "bert" preprocess mode, which applies BERT-specific tokenization and formatting to the input text data. This mode is specifically designed for BERT models and helps optimize the preprocessing step, enhancing the model's ability to understand the context and semantics of the text.

3.7.2 Model training hyperparameters

Batch Size: We chose a batch size of 6, which determines the number of training samples processed in each iteration. This value strikes a balance between training speed and memory consumption, considering the available computational resources and dataset size.

Learning Rate: We set the learning rate to 2e-5, a value recommended by Google for fine-tuning BERT models. This learning rate choice enables effective convergence during training while minimizing the risk of overshooting the optimal solution.

Number of Epochs: The model was trained for 4 epochs, meaning the entire training dataset was processed four times. This number of epochs allows the model to learn patterns and generalize well to the dataset without over-fitting.

3.7.3 Data split hyperparameters

Test Size: We partitioned the dataset into training and testing sets using a test size of 0.2 (20%). This split allocates 80% of the data for training and 20% for testing, ensuring a sufficient amount of data for evaluation while preserving a sizable training set.

By carefully selecting these Hyperparameters, including the test size, max length, preprocess mode, batch size, learning rate, and epochs, we aimed to optimize the performance of our

system for detecting SQL injections. These choices were based on prior knowledge, best practices and recommendations. The hyperparameters collectively contribute to the effectiveness and accuracy of our model in identifying SQL injection attacks.

3.8 Conclusion

In this chapter, we have described the design and implementation of a system for detecting SQL injections using deep learning. We utilized a dataset of SQL injection attacks and normal queries to train a BERT model for classification. The preprocessing steps involved converting the text data into a format suitable for BERT model input and splitting the data into training and testing sets. Our results showed that the trained model was effective in detecting SQL injection attacks with a high degree of accuracy. By sharing the details of our approach, we have provided a comprehensive understanding of how deep learning can be utilized for SQL injection detection.

Chapter 4

Test and Evaluation

4.1 Introduction

The detection of SQL injections using AI techniques, such as machine learning and deep learning algorithms, has gained the interest of many researchers in this field. These techniques have been shown to be effective at identifying SQL injection attacks with high accuracy.

In this chapter, we will discuss the test and evaluation of our model for detecting SQL injection attacks. We will use a variety of metrics, including accuracy, precision, recall, and F1 score. We will also compare the performance of our model to other machine learning algorithms and related works.

4.2 Evaluation metrics for assessing model performance

In the field of machine learning and classification tasks, evaluation metrics play a crucial role in assessing the performance of models. These metrics provide quantitative measures that help us understand the accuracy, effectiveness, and reliability of model predictions. When evaluating the performance of classification models, it is essential to examine the appropriate evaluation metrics that provide insights into their strengths and weaknesses. In this section, we will explore some of the most commonly used evaluation metrics that provide valuable insights into the performance of classification models.

4.2.1 Confusion matrix

The confusion matrix provides a tabular representation of the model's predictions against the actual labels. It allows us to visualize the distribution of true positives, true negatives, false positives, and false negatives, providing valuable insights into the model's performance [26].

	Positive Prediction	**Negative Prediction**
Positive Class	True Positive (TP)	False Negative (FN)
Negative Class	False Positive (FP)	True Negative (TN)

TP (True Positives): Correctly predicted positive instances.

TN (True Negatives): Correctly predicted negative instances.

FP (False Positives): Incorrectly predicted positive instances.

FN (False Negatives): Incorrectly predicted negative instances.

4.2.2 Accuracy

Accuracy is a commonly used evaluation metric that measures the overall correctness of model predictions. It calculates the ratio of correct predictions to the total number of instances. Accuracy provides a general overview of the model's performance across all classes [26].

```
Accuracy = Correct Predictions / Total Predictions
```

4.2.3 Precision

Precision focuses on the proportion of correctly identified positive predictions (true positives) out of the total positive predictions made by the model. It helps assess the model's ability to minimize false positives [27].

```
Precision = TruePositive / (TruePositive + FalsePositive)
```

4.2.4 Recall

Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions captured by our model out of the total actual positive instances. It reflects the model's ability to minimize false negatives [27].

```
Recall = TruePositive / (TruePositive + FalseNegative)
```

4.2.5 F1 Score

The F1 score is a combined metric that balances precision and recall. It provides a harmonic mean of these two measures and offers a comprehensive evaluation of the model's performance [28].

```
F-Measure = (2 * Precision * Recall) / (Precision + Recall)
```

By analyzing these evaluation metrics, we can gain a deeper understanding of how our classification model performs and identify areas for improvement. These metrics provide valuable insights into the model's strengths and weaknesses, allowing us to make informed decisions to enhance its performance.

4.3 Model performance analysis

In this section, we evaluate the performance of our SQL injection detection model based on the BERT architecture. As already discussed the model was trained on a dataset containing 22,599 samples, consisting of both SQL injections instances and normal SQL queries.

We assess the model's performance using a variety of evaluation metrics, including accuracy, precision, recall, F1 score, and the confusion matrix. These metrics provide valuable insights into the model's ability to correctly classify SQL injection instances and non-malicious queries.

The performance results obtained are as follows:

Accuracy: The model achieved an accuracy of **99.98%** during training and validation, indicating its high level of accuracy in classifying the queries.

F1 Score: The F1 score, which considers both precision and recall, also reached an impressive value of **99.98%.** This indicates a balance between correctly identifying SQL injection attacks (precision) and capturing all actual SQL injection instances (recall).

Precision: The precision of the model, which measures the proportion of true positive predictions out of all positive predictions, was **99.96%.** This indicates a very low rate of false positives, meaning that the model maintains a high level of confidence in its SQL injection detection predictions.

Recall: The recall of the model, which measures the proportion of true positive predictions out of all actual positive instances, was **100%.** This indicates the model's ability to capture nearly all instances of SQL injection attacks, resulting in a low rate of false negatives.

The confusion matrix provides a more detailed breakdown of the model's performance:

	Positive Prediction	**Negative Prediction**
Positive Class	2284	0
Negative Class	1	2235

The confusion matrix reveals that out of the 2284 positive instances (normal queries), the model correctly identified 2284 instances as positive (true positives). It also correctly classified 2235 out of 2236 negative instances as negative (SQL injections) while misclassifying one instance as positive (false positive).

These results are consistent with the findings of Srishti Lodha and Atharva Gundawar from the Department of Computer Science and Engineering at Vellore Institute of Technology [29], who made a similar study using the BERT architecture for SQL injection detection. Their research demonstrated comparable performance and highlighted the effectiveness of the BERT model in accurately identifying SQL injection attacks.

Overall, the performance results demonstrate the high accuracy, precision, recall, and F1 score of our SQL injection detection model. These results, in alignment with the work of Srishti Lodha and Atharva Gundawar, underscore the effectiveness of the BERT model for accurately detecting SQL injection attacks while minimizing false positives and false negatives.

4.4 Comparative analysis with other approaches

In this section, we present a comparative analysis of our BERT-based SQL injection detection model with other commonly used approaches available in the literature. While we didn't evaluate the performance of the alternative approaches ourselves, we have collected and compiled information from various sources on their reported performance. By comparing our model with these approaches, we aim to provide insights into the effectiveness of our BERT-based model for SQL injection detection. The comparison is based on commonly used evaluation metrics such as accuracy, precision, recall, and F1 score. The findings of this analysis are summarized in the following table.

Model name	Training Accuracy	Validation Accuracy	Precision	Recall	F1
KNN	100%	99.12%	98.85%	99.52%	99.18%
SVM	92.78%	92.44%	90.21%	96.36%	93.19%
BERT	**99.99%**	**99.98%**	**99.96%**	**100%**	**99.98%**

The compared approaches were trained on a dataset of approximately 42,000 data points, while our dataset consisted of 22,599 samples. This disparity arises from the fact that we obtained the original dataset from the same resource, but we had to perform data cleaning and preprocessing to ensure its quality.

The comparison shows that BERT performed the best among the compared models in terms of key evaluation metrics, including accuracy, precision, recall, and F1 score. These results

affirm the effectiveness of BERT in detecting SQL injection attacks, highlighting its superior performance in our study.

4.5 Model performance evaluation on new data

In this section, we assess the performance of our deep learning model, trained on a specific dataset, on a new dataset obtained from Kaggle called "sqli". This dataset consists of 1100 samples and serves as a valuable benchmark to evaluate our model's capabilities on unseen data.

We started by examining the overlap between the training data and the new dataset. Through careful analysis, we identified 90 common sentences shared between the two datasets. We also quantified the similarity between the training data and the new dataset, revealing a remarkable similarity score of 9.48%. This score highlights the percentage of sentences present in the training dataset that also appear in the test dataset, underscoring the model's proficiency in handling comparable contexts and retaining its predictive power across different datasets.

Ultimately, the model's performance on the new data is outstanding, boasting an accuracy of 99.73%. This accuracy metric signifies the percentage of correct predictions made by the model when tested on the new dataset. The exceptionally high accuracy demonstrates the robustness and reliability of our deep learning model in identifying SQL injections, even when confronted with previously unseen data.

4.6 Conclusion

In conclusion, our model for detecting SQL injection attacks has shown impressive effectiveness. Through testing and evaluation using various metrics, we have demonstrated its accuracy in identifying SQL injection attacks also the comparison with other models further validates its superior performance.

General Conclusion

This project has made significant contributions to the field of web application security by employing BERT for the detection of SQL injections. The primary contribution of our work lies in demonstrating the effectiveness of BERT in accurately identifying SQL injection attempts in real-time. By exploiting the contextual understanding and semantic representation capabilities of BERT, we have achieved superior performance compared to other machine learning models. Our results provide valuable insights into the application of deep learning techniques for mitigating the major risk of SQL injections on web applications.

Despite the promising results obtained, our work has certain limitations that should be admitted. Firstly, the dataset used for training and evaluation, while comprehensive, may not be large enough to fully exploit the potential of BERT. With larger datasets, BERT's ability to capture complex patterns and nuances could be further enhanced. Secondly, we acknowledge that our experiments were conducted in a controlled environment, and the model was not tested or employed in a real-world scenario.

Based on our analysis of the BERT-based model for SQL injection detection, there are several recommendations and future directions to further enhance its capabilities:

- Explore a larger and more diverse dataset. A larger and more diverse dataset will help the model to learn more about different attack scenarios and improve its performance.
- Investigate the model's performance in real-world scenarios. The model's performance in real-world scenarios should be investigated to understand its limitations. This can be done by deploying the model in a production environment and monitoring its performance.
- Continuously update the model with new attack patterns. The model should be continuously updated with new attack patterns to improve its accuracy in detecting new attacks.
- Expand the scope to detect and classify various types of cyber attacks. The scope of the model can be expanded to detect and classify various types of cyber attacks using a multi-classification model. This will provide a more comprehensive approach to threat detection and mitigation in cybersecurity.

By addressing these recommendations and future directions, we can advance the field of SQL injection detection and contribute to the development of more effective security solutions.

References

[1] OWASP, "Top 10 Web Application Security Risks," 2021. [Online]. Available: https://owasp.org/www-project-top-ten/.

[2] OWASP, "SQL Injection," 2023. [Online]. Available: https://owasp.org/www-community/attacks/SQL_Injection.

[3] Researchgate, "Web Application Security by SQL Injection DetectionTools," 2012.

[4] J. Clarke-Salt, SQL Injection Attacks and Defense, 2012.

[5] OWASP, «Blind SQL Injection,» 2023. [En ligne]. Available: https://owasp.org/www-community/attacks/Blind_SQL_Injection.

[6] S. M. P. Dafydd, The Web Application Hacker's Handbook: Finding and Exploiting Security Flaws, 2011.

[7] Positive Technologies, "how-to-prevent-sql-injection-attacks," Augest 2019. [Online]. Available: https://www.ptsecurity.com/ww-en/analytics/knowledge-base/how-to-prevent-sql-injection-attacks.

[8] T. T. R. &. F. J. Hastie, The elements of statistical learning: data mining, inference, and prediction (2nd ed.), 2009.

[9] C. M. Bishop, Pattern recognition and machine learning, 2006.

[10] R. S. &. B. A. G. Sutton, "Reinforcement learning: an introduction (2nd ed.)," in MIT Press, 2018.

[11] I. B. Y. &. C. A. Goodfellow, "Deep learning," in MIT Press, 2016.

[12] M. A. Nielsen, "Neural Networks and Deep Learning," in Determination Press, 2015.

[13] A. Graves, "Generating Sequences With Recurrent Neural Networks," 2013. [Online]. Available: https://ui.adsabs.harvard.edu/abs/2013arXiv1308.0850G/abstract.

[14] J. G. C. C. K. &. B. Y. Chung, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014. [Online]. Available: https://nyuscholars.nyu.edu/en/publications/empirical-evaluation-of-gated-recurrent-neural-networks-on-sequen.

[15] S. &. S. J. Hochreiter, "Long short-term memory. Neural computation.," MIT Press, 1997.

[16] A. Graves, Supervised sequence labelling with recurrent neural networks, 2012.

[17] K. V. M. B. G. C. B. D. B. F. S. H. &. B. Y. Cho, Learning phrase representations using RNN encoder-decoder for statistical machine translation, 2014.

[18] A. S. N. P. N. U. J. J. L. G. A. N. .. &. P. I. Vaswani, "Attention is all you need," 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[19] M.-W. C. K. L. a. K. T. Jacob Devlin, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Volume 1 (Long and Short Papers), 2019.

[20] H. Z. X. H. a. J. Z. Yan Zhang, Benchmarking Neural Network Models for SQL Injection Detection, 2020.

[21] A. Géron, Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, 2019.

[22] S. &. M. V. Raschka, Python Machine Learning: Machine Learning and Deep Learning with Python, Packt Publishing, 2019.

[23] T. R.-K. B. P. F. G. B. E. B. M. F. J. Kluyver, Jupyter Notebooks—a publishing format for reproducible computational workflows, In ELPUB, 2016.

[24] A. T. A. &. H. J. D. Rule, Rapid prototyping interactive data visualizations with Jupyter notebooks, IEEE transactions on visualization and computer graphics, 2018.

[25] "Kaggle," 2023. [Online]. Available: https://www.kaggle.com/datasets/syedsaqlainhussain/sql-injection-dataset.

[26] M. &. L. G. Sokolova, A systematic analysis of performance measures for classification tasks, 2009.

[27] D. M. Powers, "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation," Journal of Machine Learning Technologies, 2011.

[28] C. J. Van Rijsbergen, Information Retrieval (2nd ed), 1979.

[29] S. L. a. A. Gundawar, SQL Injection and Its Detection Using Machine Learning Algorithms, 2023.