Arab American University
Faculty of Graduate Studies
Department of Natural, Engineering and Technology Sciences
Master Program in Cyber Security

Comparative Analysis of Various Algorithms for Detecting Sql Injection in Web Applications

Aysar Abd Al-Raheem Meflah Al-Weshahi
202112892

Supervision Committee:
Dr. Rami Hadrob
Dr. Mahmoud Obaid
Dr. Amjad Al-Ratrout
Dr. Rami Youssef

This Thesis Was Submitted in Partial Fulfilment of the Requirements for the Master Degree in Cybersecurity.

Palestine, Feb/2025

Arab American University

Faculty of Graduate Studies

Department of Natural, Engineering and Technology Sciences

Master Program in Cyber Security

Thesis Approval

Comparative Analysis of Various Algorithms for Detecting Sql Injection in Web Applications

Aysar Abd Al-Raheem Meflah Al-Weshahi

202112892

This thesis was defended successfully on 6/2/2025 and approved by:

Thesis Committee Members:

Name	Title	Signature
1. Dr. Rami Hadrob	Main Supervisor	
2. Dr. Mahmoud Obaid	Co-Supervisor	
3. Dr. Amjad Al-Ratrout	Member of Supervision Committee	Dr Amjad RATTROUT
4. Dr. Rami Youssef	Member of Supervision Committee	

Palestine, Feb/2025

Declaration

I declare that, except where explicit reference is made to the contribution of others, this thesis is substantially my own work and has not been submitted for any other degree at the Arab American University or any other institution.

Student Name: Aysar Abd Al-Raheem Meflah Al-Weshahi
Student ID: 202112892
Signature: Aysar Weshahi
Date of Submitting the Final Version of the Thesis: 1.7.2025

Dedication

I dedicate this work to my dear parents, whose constant support, prayers and encouragement have been my guiding light throughout this journey. I thank my family and friends for standing by my side. Finally, I thank my supervisors whose guidance and advice have been invaluable.

Student’s Name: Aysar Abd Al-Raheem Meflah Al-Weshahi

Acknowledgments

First and foremost, I dedicate my work to the courageous prisoners whose unwavering determination motivates us to keep fighting for justice and freedom, as well as to our honorable martyrs who gave their lives in defense of Palestine. You will always be in our hearts.

I want to sincerely thank my distinguished supervisors, Dr. Rami Hadrob and Dr. Mahmoud Obaid, for their crucial advice and assistance during this research trip. Their advice was really helpful to me in conquering obstacles and accomplishing my objectives.

My foundation has been the love of my mother, my inspiration and source of strength, and my dear dead father, my role model. I want to express my gratitude to my spouse, daughter, brothers, and sisters for their unending support and company.

I am also deeply thankful to my family and colleagues whose continuous support has been a constant source of strength.

Finally, I sincerely hope for peace and well-being for all those affected by conflict in Gaza, the West Bank, and beyond.

Title: Comparative Analysis of Various Algorithms for Detecting Sql Injection in Web Applications.

Student's Name: Aysar Abd Al-Raheem Meflah Al-Weshahi

Supervision Committee:
1.Dr. Rami Hadrob
2. Dr. Mahmoud Obaid
3. Dr. Amjad Al-Ratrout
4. Dr. Rami Youssef

Abstract

Injection attacks, especially SQL injection, remain one of the biggest threats to online application security, compromising user privacy, system integrity, and data confidentiality(Farooq, 2021). This paper compares machine learning (ML), deep learning (DL), and hybrid models for detecting such attacks. In addition to traditional models such as logistic regression, naive Bayes, decision trees, and random forests, a variety of algorithms were evaluated, including more complex approaches such as forward neural networks (FFNNs), convolutional neural networks (CNNs), and hybrid combinations such as FFNN+Naive Bayes or FFNN+SVM (Liu et al., 2021).

Two different datasets were used to evaluate the models using three popular text-to-vectorization techniques: Count Vectorizer, Word2Vec, and TF-IDF. Evaluation metrics such as execution time, accuracy, precision, recall, and F1 score were used to provide a comprehensive assessment (Yacouby & Axman, 2020). The comes about appears that FFNNs embedded with Word2Vec are the deep learning models with the highest accuracy (up to 98.9%) and moderately quick execution times. Hybrid models such as FFNN + Naive Bayes and NN + SVM also illustrated prevalent execution (98.1%) when utilizing Count Vectorizer, combining the qualities of both deep learning and traditional machine learning.

In contrast, traditional models such as Naive Bayes and Decision Trees struggled to represent more complex or dense vectors, especially with TF-IDF and Word2Vec. XGBoost emerged as the best performer among traditional models using TF-IDF, achieving 98.4% accuracy but with a longer execution time. Despite its accuracy, SVMs suffer from scalability issues due to their high computational cost on high-dimensional data.

This study demonstrates that while traditional models may perform well for applications requiring fast completion, deep learning and hybrid techniques outperform when accuracy is a major concern. The results encourage further research on real-time deployment frameworks for SQL injection detection, which also supports the wider use of hybrid models that combine comprehensive feature extraction and efficient classification.
Keywords: SQL Injection, Machine Learning, Deep Learning, Hybrid Models.

Table of Contents

List of Tables

List of Figures

List of Definitions of Abbreviations

- Acc: Accuracy
- ANN: Artificial Neural Networks
- API: Application Programming Interface
- Bi-LSTM: Bidirectional Long Short-Term Memory
- CNN: Convolutional Neural Network
- DL: Deep Learning
- DT: Decision Tree
- IDF: Inverse Document Frequency
- FFNN: Feedforward Neural Network
- FN: False Negative
- FP: False Positive
- LR: Logistic Regression
- LSTM: Long Short-Term Memory
- ML: Machine Learning
- NB: Naïve Bayes
- NN: Neural Network
- NLP: Natural Language Processing
- PCA: Principal Component Analysis
- PNN: Probabilistic Neural Network
- RF: Random Forest
- RBAC: Role-Based Access Control
- RCE: Remote Code Execution
- RDBMS: Relational Database Management Systems
- RNN: Recurrent Neural Network
- SQL: Structured Query Language
- SQLI: SQL Injection
- SQLIAs: SQL injection attacks
- SVM: Support Vector Machine
- TF-IDF: Term Frequency–Inverse Document Frequency
- TN: True Negative
- TP: True Positive
- WAFs: Web Application Firewalls

- XGBoost: Extreme Gradient Boosting
- XSS: Cross-Site Scripting

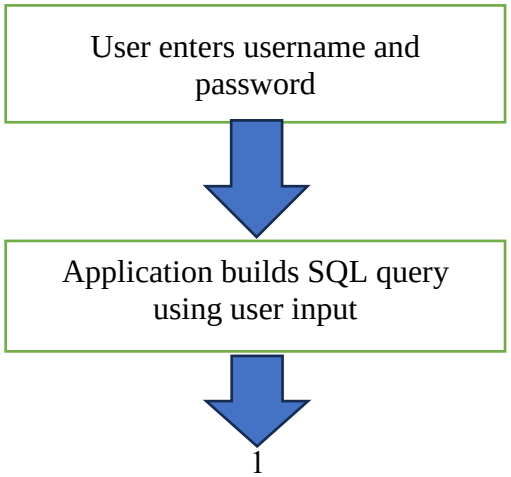
1. Chapter One: Introduction

1.1 Background

Within the modern computerized period, web-based applications are significant to the day-to-day operations and exchanges of numerous firms. These apps utilize Relational Database Administration Frameworks (RDBMS), which are fueled by Structured Query Language (SQL). Be that as it may, the potential for SQL injection attacks (SQLIAs), which are activated by the use of SQL, could be a major issue for organized security experts.

A 2023 report states that SQL injection attacks are responsible for around 23% of significant web application vulnerabilities globally(Alanda et al., 2021; Farooq, 2020) . Additionally, SQL injection attacks came in third place on OWASP's ranking of the top 10 threats to web application security in 2021. This vulnerability continues to be a serious threat to data security even after it was identified more than 20 years ago. SQL injection threats allow attackers to insert malicious code into input fields, alter SQL questions, and even access or alter data without permission (Flores Jr & Monreal, 2024).

SQL injection (SQLI) is a well-known attack technique that modifies a backend database and grants unauthorized access to data by using malicious SQL code. This flaw gives hackers the ability to circumvent web application limitations and authentication processes, which could result in a database breach. The steps in a SQL injection attack are shown in Figure 1.



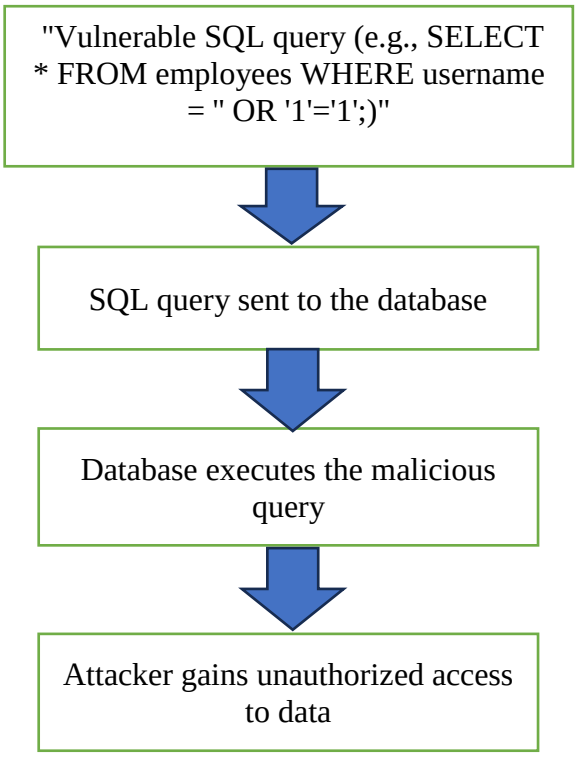


Figure 1.1 : SQL Injection Attack

A few significant cyber incidents in recent years have highlighted the fundamental threat posed by SQL injection attacks. For example, in 2020, an inadequately secured database of client qualifications was exploited using SQL injection techniques, resulting in a breach involving more than 200 million records (Alenezi et al., 2021; Ibarra-Fiallos et al., 2021). These incidents not only damage firms' reputations and compromise sensitive information, but they can result in notable financial setbacks and legal outcomes. In addition, advancements in automated attack tools have enabled non-professional programmers to launch contemporary SQL injection attacks with minimal effort (Alghawazi et al., 2022; Deriba et al., 2022).

It is critical for web architects to be mindful that assailants may utilize despicably cleaned client input. An assortment of SQL injection attacks, such as blind SQL injection, union-based attacks, padding attacks, and error-based attacks, can emerge from improper taking care of client input. For instance, a padding attack circumvents validation tests by changing SQL query conditions so they always evaluate to true. Union-based attacks use a UNION statement to commit illicit acts, while error-based attacks exploit server failures

to induce database structure. Blind SQL injection uses true-false queries or time delays to extract data (Demilie & Deriba, 2022).

Additionally, SQL injection attacks have become increasingly sophisticated, and attackers are now combining these tactics with other types of vulnerabilities, including remote code execution (RCE) or cross-site scripting (XSS), to increase their impact (AL-Maliki & Jasim, 2022).

SQLIA frequently results in unauthorized logins, which get around verification procedures by using stuffing attacks to obtain sensitive data. It is possible to handle questions like INSERT that appear blamelessly incorrectly. The risks of SQL injection are highlighted by a number of examples, including as the Sony PlayStation Organization breach, which resulted in billions of dollars in damages and the loss of millions of accounts (Su et al., 2018).

Researchers found that over 15% of SQL injection attacks were able to get past conventional Web Application Firewalls (WAFs) in 2021 by taking advantage of their dependence on preset rules rather than adaptive learning models(Ibarra-Fiallos et al., 2021).

Even conventional security measures like Web Application Firewalls (WAFs) frequently fail to detect, identify, or prevent all types of SQLIA, despite the fact that several attempts have been made to mitigate or reduce these threats and assaults. In order to solve these issues, researchers have turned to artificial intelligence techniques. In order to detect SQL injection attacks, this study suggested a number of techniques and selected the most accurate and least time-consuming one (Deriba et al., 2022).

1.2 Problem Statement and Research Questions

In spite of the appropriation of standard security measures such as input approval and web application firewalls (WAFs), SQL injection (SQLi) assaults stay a basic risk to the privacy, astuteness, and accessibility of web applications. These ordinary resistances regularly come up short to identify advanced or muddled assault designs, taking off cutting edge frameworks uncovered to serious vulnerabilities(Farooq, 2021).

Later advancements in machine learning (ML) and deep learning (DL) offer promising capabilities in identifying and anticipating SQLi assaults by learning complex behaviors and adjusting to advancing dangers. Be that as it may, the usage of these methods presents a few key challenges:

- Imbalanced datasets that predisposition models toward generous inquiries.
- Limited generalization, where models battle to identify novel or real-world assault variations.
- The black-box nature of DL models, which limits their interpretability and auditability.
- Evasion techniques used by attackers to bypass intelligent detection systems.
- High computational costs and latency, especially in deep models, influencing real-time appropriateness.
- Integration complexity in hybrid models, making tuning and optimization troublesome.
- Scarcity of realistic and up-to-date datasets reflecting current threat landscapes.
- Inconsistency with advanced web innovations, such as energetic inquiry era in Precise or Respond.
- Need of standardized assessment criteria, ruining reasonable comparison between discovery approaches.

These challenges highlight the critical require for more shrewdly, proficient, and versatile SQLi discovery frameworks able of working in assorted situations whereas standing up to avoidance endeavors(Abdullah & Abdulazeez, 2024; Xia et al., 2024).

Appropriately, this investigate explores the utilize of ML and DL methods to improve SQLi location and looks for to address the taking after inquire about questions:

1. What are the foremost viable machine learning procedures for recognizing SQL injection assaults?
2. Which methods—deep learning or conventional machine learning—are more capable in distinguishing distinctive sorts of SQLi assaults?
3. What are the trade-offs between computational taken a toll, exactness, and effectiveness?
4. What challenges must be tended to in planning adaptable and dependable brilliantly location frameworks?
5. How can cross breed models be optimized to use the qualities of numerous calculations?
6. What part does include designing (e.g., word vectorization) play in moving forward discovery exactness?
7. To what degree can real-time or low-latency location be accomplished without relinquishing execution?
8. Which model achieves higher detection accuracy for SQLi attacks: ML or DL or hybrid algorithms?

1.3 Significance of the Study

The main goal of the study is to ascertain if deep learning and traditional algorithms are more effective at identifying SQL injections. This will allow for the development of strategies and tactics to defend apps against SQL injection attacks.

A few groups in the actual world can profit from the study's conclusions. Through the use of CI/CD pipelines or well-known online systems, engineers can leverage the information acquired to specifically incorporate tighter security measures into their development processes.

Organizations may lower the risk of breaches, safeguard private client information, and comply with data assurance regulations by putting these improved

detection strategies into practice. Ultimately, this study benefits customers by ensuring that their personal information is better protected in today's increasingly digital environment.

1.4 Objectives of this Study

The main objectives of this study are to:

1. Assess and compare the viability of traditional machine learning and deep learning algorithms in detecting SQL injection attacks.
2. Look at the execution of these calculations over different datasets to guarantee their generalizability.
3. Explore the trade-offs between accuracy, execution time, and computational productivity.
4. Analyze the effect of highlight representation procedures (e.g., Count Vectorizer, TF-IDF, Word2Vec) on discovery precision.
5. Investigate the achievability of crossover models that combine the qualities of different calculations.
6. Assess the execution of the models utilizing different measurements, including accuracy, precision, recall, F1-score, and execution time.
7. Distinguish the challenges and impediments in conveying real-time and versatile SQL infusion discovery frameworks.
8. Give viable proposals for making strides current discovery procedures based on the test discoveries.

1.5 Thesis Organization

To ensure a coherent and coherent flow of ideas, this proposal has been effectively divided into discrete portions. From the fundamental ideas to the analysis, application, and outcomes, each section focuses on a distinct facet of the investigation. The framework seeks to highlight important discoveries and useful insights while offering a thorough grasp of the research issue. It is as follows:

- Chapter I Introduction:

  This section discusses SQL injection and provides pertinent cost and resultant damage figures. Additionally, it outlines the goals, significance, and research problem.
- Chapter II Literature Review:

  This section offers a thorough analysis of a number of algorithms, going over their characteristics, accuracy rates across various studies, advantages, and disadvantages.
- CHAPTER III Background and methodology:

  The approach used in the study is explained in this section. It gives background details on the many kinds of SQL injection attack targets and origins. Along with other important metrics, it also provides an illustration of how evaluation metrics like precision, F1 Score, recall, and precision are computed.
- CHAPTER IV Model Design:

  This section focuses on obtaining datasets, using the code to evaluate parameter values, and choosing the best possible configurations.
- CHAPTER V Results and Evaluation:

  This section displays the accuracy results for each classification technique (Count Vectorizer, Word Vectorizer and TF-IDF) across the datasets. It evaluates the feasibility, efficacy, and deployment time of each algorithm to determine the most successful and adaptable approach.
- CHAPTER VI Conclusions:

  The key distinctions between the algorithms are covered in this section, along with the reasons why some work better than others. On the basis of the results, it also offers a number of recommendations.
- CHAPTER VII References:

  To provide thorough and accurate documentation, the last part lists all of the literary sources used during the investigation.

2. Chapter Two: Literature Review

2.1 Overview of Algorithms

This chapter examines a number of studies and publications on traditional machine learning and deep learning methods for detecting SQL injection (SQLI) attacks. Since this research paper will cover a wide range of methods, including traditional algorithms, machine learning algorithms, feature extraction methods like Count Vectorizer and Word Vectorizer, and how they affect each algorithm's accuracy, as well as manipulating the values of variables or coefficients for each algorithm and their effect on its level of accuracy, the primary goal of studying this literature is to confirm the effectiveness of algorithms, select the best ones to incorporate into the research, and identify knowledge gaps within these studies and try to minimize them as much as possible (Abebe et al., 2024).

2.2 Traditional Machine Learning Algorithms

1-Naive Bayes:

Naive Bayes with an accuracy rate of 93.3 percent, it is a powerful and effective method for detecting SQL injection attacks. It is even more effective when role-based access control (RBAC) is added. Because it relies on the conditional independence assumption, it performs mediocrely well when complex features are present. This technique is more successful at identifying injection risks when combined with role-based access control (RBAC) (Arnap, 2024; Demilie & Deriba, 2022; Hasan et al., 2019).

Strengths:

1. Accuracy: With an accuracy record of 95.59 percent , it has demonstrated its strength (Pramono et al., 2024).
2. Simplicity and speed: This algorithm's speed is a strength because it takes very little time. Its capacity to analyze data is also strengthened by the data's simplicity (is this a strength or weakness for complicated data?).

3. Scalability: Due to its high and efficient computational efficiency when compared to its equivalents from other algorithms, this approach works well with large datasets.

Weaknesses:

1. Assumption of independence of features: This algorithm's assumption that all features are independent, which results in some credibility or confidence, is one of its most significant drawbacks. It also has an adverse effect on accuracy and the dependencies of complicated features.
2. Imbalanced data: Due to their underrepresentation, this algorithm struggles to handle imbalanced data sets, which will reduce its accuracy (Arnap, 2024).
3. Dealing with complex relationships: This algorithm's low accuracy rate in complicated feature connections is caused by its inability to handle them.

Despite several challenges, this computation is still a successful and efficient option for detecting SQL injection attacks, especially in large-scale and real-time applications. It is lucrative in cybersecurity since its flaws can often be fixed with complimentary techniques (Jemal et al., 2020).

2-SVMs (support vector machines):

The performance of support vector machines (SVMs) has continuously surpassed that of many conventional methods, particularly when paired with hybrid approaches. With an accuracy rate of almost 99 percent, they are a great option for SQL injection (SQLI) detection due to their capacity to handle high-dimensional data. Their computational complexity is still a significant disadvantage despite their exceptional efficacy (Krishnan et al., 2021; Peralta-Garcia et al., 2024).

Strengths:

1. Ability to handle complex data: This approach is appropriate for identifying SQL injection attacks that include high-dimensional data because it can handle complex, high-dimensional data properly and effectively.

2. Accuracy: It has demonstrated outstanding performance and very high accuracy, detecting SQL injections with an accuracy of up to 99 percent (Muslihi & Alghazzawi, 2020).

Weaknesses:

1. Computational complexity: Even though this approach is efficient and effective, it is unsuccessful when used in real-time systems or with huge data sets since it demands a lot of processing resources, which can be costly. Even though this approach is efficient and effective, it is unsuccessful when used in real-time systems or with huge data sets since it demands a lot of processing resources, which can be costly.
2. Challenges in choosing the kernel: SVMs' performance is mostly dependent on selecting the right kernel function, which might be challenging in some situations and ultimately result in failure.
3. Sensitivity to unbalanced data: This algorithm may show relatively low accuracy due to its handling of an unbalanced data set, where certain categories are not adequately represented(Demilie & Deriba, 2022).

3-Logistic Regression:

Because it is easy to use and understand, calculated logistic regression is used as a pattern in execution assessments. In any event, its inability to detect nonlinear designs limits its ability to identify complicated SQL Injection (SQLI) threats.

Strengths:

1. Effortlessness and Interpretability:
   Logistic Regression is clear to execute, and its outcomes are simple to translate, making it a well-known choice for starting modeling.
2. Standard Execution: It serves as a strong pattern for comparing more complex models in SQLI discovery tasks.

Weaknesses:

1. Limited to Linear Relationships: Its ability to detect intricate nonlinear SQLI designs is limited by calculated logistic regression, which shows a direct correlation between the independent elements and the log chances of the dependent variable.
2. Lower Accuracy in Complex Scenarios: It has been suggested that, in comparison to more sophisticated models, logistic regression achieves poorer accuracy. For instance, while detecting SQLI attacks Although models such as Nave Bayes get up to 93.3 percent accuracy, logistic regression achieved a precision of 95.1 percent (Alghawazi et al., 2022).
3. Sensitivity to Multicollinearity: Highly linked features have the potential to negatively impact Logistic Regression's effectiveness and produce estimates that are not trustworthy.

4-Decision Trees:

Decision trees outperform the J48 method on simple classification tasks, although they struggle a bit on more complicated datasets. Although they have achieved a 96 percent accuracy rate, overfitting techniques such irregular forests and slope boosting have been introduced to get over these restrictions, primarily due to advancements in their application (Sharma et al., 2020).

Strengths of Decision Trees:

1. Effortlessness and interpretability: Decision trees are simple to understand and visualize, making them valuable for investigating crude information and clarifying outcomes.
2. Quick preparation and forecast: They require relatively few computational assets, permitting for fast demonstration preparation and real-time expectations.

Weaknesses of Decision Trees:

1. Overfitting: This algorithm can become very complex, so it is more likely to pick up noise in the data, leading to overfitting and poor generalization to new data.
2. Instability: A small change in the data can significantly affect the tree structures and even cause unnecessary and complex structures to become unhelpful, affecting the results(Le et al., 2024; Peralta-Garcia et al., 2024).

5-Passive Aggressive Classifier:

This technique is distinguished by its real-time SQL injection detection capabilities, which enable it to react efficiently by promptly changing its view when mistakes arise. After converting SQL queries into discrete vectors, this algorithm categorizes them based on the previously gathered processing data. A 79 percent accuracy rate was attained (Krishnan et al., 2021).

Strengths:

1. Real-time detection: The design of this algorithm allows it to identify SQL injection attacks as they occur, making it robust and suitable for applications that require immediate threat detection.
2. Efficient model updates: When the model is updated upon the occurrence of misclassifications, this greatly helps in adapting to new patterns without the need to retrain the model.

Weaknesses:

1. Its accuracy: It may not be as reliable as previous algorithms (SVM, Naïve Bayes).
2. Limited capacity: This algorithm has limited capacity for complex patterns, as the passive-aggressive classifier may have difficulty capturing complex nonlinear relationships within the data, which may lead to misclassifications.

In summary, this calculation's strengths lie in real-time discovery and fruitful demonstration, but its limitations in handling complicated designs suggest that it would work better when combined with other tactics or integrated into a collection system for SQLI detection (Krishnan et al., 2021).

6-Gradient Boosting Classifier:

Through rigorous training on labelled SQL query datasets, this system achieves a high 99 percent accuracy rate, making it a dependable option for identifying SQL injection threats. It does this by utilizing numerous decision trees to uncover complicated patterns in data (Farooq, 2021).

Strengths:

1. Accuracy: This algorithm has high accuracy due to its ability to combine multiple decision trees, enabling it to capture complex data patterns.
2. Flexibility: It can be added with other algorithms (hybrid), as it is effective in classification and regression tasks.
3. Feature importance: The ability to effectively determine the importance of a feature, thus helping to understand which variables contribute most to predictions.

Weaknesses:

1. Computational complexity: This model can take a lot of time to train and requires significant resources.
2. Risk of overfitting: If the data is not processed properly, it may limit its ability to generalize to new, unselected data.
3. Parameter sensitivity: This algorithm relies heavily on hyperparameter settings, requiring fine-tuning to obtain optimal and robust results.

In conclusion, gradient boosting is a useful technique for identifying SQL injection attacks, improving accuracy, and demonstrating intricate data relationships; however, it necessitates significant computational resources and requirements, and its sensitivity to parameters necessitates careful use (Le et al., 2024; Roy et al., 2022).

7-AdaBoost Classifier:

This method is eminent for its extraordinary capacity to identify SQL injection dangers and for combining less pertinent classifiers into a single, exceedingly exact representation by finding concealed patterns within the information. With an accuracy rate of nearly 99 percent, this framework is particularly powerful in identifying false questions, which will beat customary strategies(Le et al., 2024). As a result, its flexibility makes it a profitable cybersecurity device. AdaBoost is still a flexible and easy-to-use arrangement, in any case of how clear or complex SQL injection attack plans are, especially as attack strategies continue to advance(Farooq, 2021).

Strengths:

1. Accuracy: This algorithm boosts the performance of weak classifiers, thus giving a robust model capable of detecting SQL injection attacks.
2. Adaptability: The algorithm focuses on hard-to-classify cases by adjusting the weights, which improves its ability to detect complex SQLI patterns.
3. Flexibility: AdaBoost can be combined with multiple base learners, allowing for customization based on the characteristics of a specific dataset.

Weaknesses:

1. Sensitivity to noisy data: AdaBoost can overemphasize misclassified occurrences, leading to overfitting, particularly in the presence of noise, and hence unnaturally decreasing exactness.
2. Computational Complexity: The iterative nature of the calculation can increase computational time, particularly with huge datasets, and there can be an asset weight.
3. Diminishing Returns: After a certain number of epochs, extra boosting may abdicate negligible advancements, requiring cautious tuning(Zivkovic et al., 2024).

Table 2.1: Summary of Strengths and Weaknesses of Traditional Machine Learning Algorithms

The taking after table presents a comparative outline of the foremost commonly utilized traditional machine learning calculations within the setting of SQL infusion discovery. It traces the key qualities that make each calculation appropriate for specific scenarios, as well as their shortcomings which will constrain their execution or pertinence.

Table 2.1 : Summary of Strengths and Weaknesses of Traditional Machine Learning Algorithms

Algorithm	**Strengths**	**Weaknesses**
Naive Bayes	- Accuracy: Demonstrated up to 95.59% accuracy. - Simplicity and speed. - Scalability: Efficient with large datasets.	- Assumes independence of features. - Struggles with imbalanced data. - Poor performance with complex feature relationships.
SVM	- Handles complex, high-dimensional data. - Very high accuracy: up to 99%.	- High computational complexity. - Performance heavily depends on kernel selection. - Sensitive to unbalanced data.
Logistic Regression	- Simplicity and interpretability. - Provides a solid baseline for comparisons.	- Limited to linear relationships. - Lower accuracy in complex scenarios. - Sensitive to multicollinearity.
Decision Trees	- Simplicity and easy visualization. - Fast training and prediction.	- Prone to overfitting. - Instability: sensitive to small changes in data.
Passive Aggressive	- Real-time detection capability. - Efficient model updates on misclassifications.	- Lower accuracy compared to other models. - Limited in handling complex nonlinear relationships.

Gradient Boosting	- High accuracy. - Flexibility (can be hybridized). - Ability to identify feature importance.	- High computational complexity. - Risk of overfitting if data is not preprocessed properly. - Sensitive to hyperparameter settings.
AdaBoost	- Boosts weak classifiers to create a strong model. - Focuses on hard-to-classify cases. - Flexible with various base learners.	- Sensitive to noisy data (may overfit). - Increased computational time with large datasets. - Diminishing returns after many iterations.

2.3 Deep Learning Models

The machine learning handle has been changed by deep learning algorithms, which have amazingly high accuracy due to their capacity to effectively handle tremendous volumes of information and assess complicated plans. They will play a critical part in this consideration, especially since they can be coordinated with crossover calculations and include extraction methods, and the larger part of calculations have precision rates over 98 percent. CNNs, RNNs, and FFNNs are among these algorithms; in any case, they have issues with computational complexity, data reliance, and the plausibility of overfitting to optimize their practicality in cybersecurity applications.

To fully use it in cybersecurity applications, issues like computational complexity, data reliance, and overfitting hazards must be resolved, notwithstanding its advantages like accuracy and adaptability.

The following sections provide a thorough examination of the leading deep learning models used for SQLI location, along with an emphasis on their salient features and limitations (Muslihi & Alghazzawi, 2020; Roy et al., 2022).

1-Convolutional Neural Networks (CNNs):

CNNs are particularly effective at identifying SQL infusion SQLI assaults because they can extract data from literary input with surprisingly high accuracy. Research shows that CNN models can achieve up to 98 percent accuracy. Their performance is in fact far better when paired with half-breed models like CNN Bi-LSTM or multilayer discernments. One example showed that a CNN-based method achieved a remarkable 99.5 percent precision, surpassing standard rule-based solutions. Additionally, half breed CNN Bi-LSTM models have achieved up to 98 percent precision in a variety of metrics, including review and F1 score correctness (Chen et al., 2021; Luo et al., 2019; Sheykhkanloo, 2017).

Strengths:

1. High Accuracy: CNNs have accomplished accuracy rates as high as 99.66% in identifying SQLI assaults, outflanking conventional rule-based strategies.
2. Feature Extraction: Their capacity to consequently extract and learn various-level highlights from input information empowers CNNs to identify complex designs related to SQLI assaults.
3. Integration with Hybrid Models: Combining CNNs with models like Bi-LSTM upgrades execution, accomplishing noteworthy precision advancements.

Weaknesses:

1. Computational complexity: In addition to a huge amount of time and computer power, this approach demands a great number of resources to train, particularly when dealing with large datasets.
2. Data dependence: The quality and amount of the data determine how effective this algorithm is, and any flaw in either of these factors can lower its efficacy.
3. Risks of overfitting: The training data may become overfitted if it is not properly arranged and organized, which would limit its capacity to generalize to new, untested data.

To appropriately utilize the computing assets in cybersecurity applications, components like overfitting and data quality must be considered, indeed, on the off chance that this strategy is solid and proficient at precisely identifying infusion attacks and extracting strong highlights(Luo et al., 2019).

2-Recurrent Neural Networks (RNNs) and Long Short-Term Memory Networks (LSTMs):

As solid, successive design learning models, RNNs and LSTMs are too valuable for distinguishing SQL infusion dangers. This algorithm's precision is incredible, according to a few things about RNN and LSTM have both achieved 94 and 96 percent precision, individually(Alghawazi et al., 2022, 2023; Reddy & Rudra, 2021; Sherstinsky, 2020).

Strengths:

1. Processing sequential data: Both algorithms are characterized by preparing successive data; in this way, they are viable in understanding the complete setting of questions.
2. Handling complex patterns: hey can distinguish complex designs in information, making them exceptionally effective in distinguishing complex SQL injection attacks.
3. Flexibility: Both algorithms are broadly utilized in different applications, counting cybersecurity and data analysis, due to their different applications.

Weaknesses:

1. Computational complexity: Such models require huge computational assets, which leads to expanded training and induction time.
2. Data requirements: Both calculations require huge datasets for viable training, which can be troublesome in different circumstances or scenarios.
3. Affect: Disgraceful tuning or overfitting can debase the execution and adequacy of the demonstrate, which can influence generalization on unused information.

RNNs and LSTMs are, in outline, viable apparatuses for recognizing SQLI attacks due to their high accuracy and capacity to handle consecutive information. Be that as it may, to ensure ideal execution, considerations like as computing complexity, time, and the requirement for huge datasets must be made(Alghawazi et al., 2023).

3-Feed-Forward Neural Networks (FFNNs):

This approach has demonstrated exceptional efficacy in identifying SQL injection attempts, especially when paired with sophisticated extraction techniques like word-to-vector conversion. With accuracy rates of up to 98.04 percent, their organized architecture allows them to identify intricate patterns in data; yet, their computational complexity has an impact on resources (Tang et al., 2020; Xin et al., 2018).

Strengths:

1. Accuracy: They have accomplished an ultra-high precision rate of up to 99%.
2. Pattern recognition: The capacity of this calculation to recognize complex designs inside information enables the compelling detection of SQL injection attacks.
3. Scalability: They can handle huge datasets effectively, making them appropriate for real-time assault location in comprehensive databases.

Weaknesses:

1. 1. Computational complexity: Preparing the algorithm can require critical assets and, thus, critical computational control and time, particularly when managing huge datasets.
2. Data dependence: The performance of the algorithm depends intensely on the quality and unwavering quality of the information, which incredibly influences its accuracy and effectiveness.
3. Risks of overfitting: On the off chance that there's no appropriate organization, the calculation may overfit the preparing data and, in this way, diminish its generalizability to unseen and concealed information.

In conclusion, our algorithm's high accuracy and robust pattern identification make it a potent and successful method for identifying SQL injection assaults. Large computational resources, overfitting, data quality, and processing methods are some of its drawbacks that prevent it from reaching its full potential in cybersecurity applications (Hassan et al., 2021).

4-Neural Network (NN):

Numerous tests have demonstrated the effectiveness and power of this method, which can be developed or hybridized since it detects SQL injection assaults with a high accuracy of 98 percent, which is an extremely high proportion(Tang et al., 2020).

Strengths:

1. Accuracy: They have achieved a high and excellent accuracy of 98%, making them highly reliable in identifying SQL injection attacks.
2. Adaptability: They have the ability to adapt to and detect different attack patterns, thus enhancing their effectiveness over time.
3. Scalability: They can handle large datasets efficiently, making them suitable for real-time detection in comprehensive and real databases.

Weaknesses:

1. Computational complexity: Training the algorithm can require significant resources and, therefore, significant computational power and time, especially when dealing with large datasets.
2. Data dependence: The performance of the algorithm depends heavily on the quality and reliability of the data, which greatly affects its accuracy and effectiveness.

   3. Risks of overfitting: If there is no proper organization, the algorithm may overfit the training data and thus reduce its generalizability to new and unseen data.

All things considered, our technique provides an unmatched level of accuracy and efficiency in detecting SQL injection threats. To fully utilize its potential in

cybersecurity applications, however, factors pertaining to computational resources, data quality, and overfitting need to be taken into account (Liu et al., 2017; Zhang et al., 2022).

5-Artificial Neural Networks (ANNs):

For SQLI discovery, a crossbreed system that combines ANNs and measurable procedures has appeared with incredible unwavering quality. These models illustrate their capacity to proficiently handle complicated assault designs by classifying malevolent demands with an exactness of 99% (Liu et al., 2017; Reddy & Rudra, 2021).

6-Natural Language Processing (NLP) with Deep Learning:

NLP and deep learning together provide an exceptional way to differentiate SQLI. These tactics focus on interpreting HTTP requests and reducing noise, which is completely different from traditional rule-based strategies. With an accuracy of up to 98.25 percent, this method has demonstrated significant promise in removing false alerts (Chen et al., 2021).

7-Probabilistic Neural Networks (PNNs):

PNNs perform exceptionally well with the use of optimization techniques, such as the BAT computation. Their 98 percent accuracy results across a range of testing situations and information components exceed the limitations of traditional localization strategies (Alarfaj & Khan, 2023).

Based on the already talked about deep learning models utilized for SQL injection detection, the taking after table presents a comprehensive comparison summarizing the most qualities and shortcomings of each algorithm. This table points to supply a clearer understanding of the key contrasts between the models and to help analysts and specialists in selecting the foremost reasonable approach depending on framework necessities, craved exactness, versatility, and specialized challenges such as computational complexity or data dependency. Table: Summary of Strengths and Weaknesses of deep learning models.

Table 2.2:Summary of Strengths and Weaknesses of deep learning models.

Algorithm	**Strengths**	**Weaknesses**
CNN (Convolutional Neural Network)	- High accuracy (up to 99.66%) - Excellent feature extraction - Works well in hybrid models (e.g., CNN-Bi-LSTM)	- High computational complexity - Strong data dependence - Risk of overfitting
RNN / LSTM (Recurrent / Long Short-Term Memory Networks)	- Good for sequential data - Effective in detecting complex patterns - Flexible for multiple applications	- Requires large datasets - Computationally intensive - Sensitive to tuning and risk of overfitting
FFNN (Feedforward Neural Network)	- Accuracy up to 99% - Strong in pattern recognition - Scalable to large datasets	- Demands high computational power - Dependent on data quality - Prone to overfitting without proper control
NN (Neural Network)	- High accuracy (up to 98%) - Adaptive to various attack patterns - Scalable and suitable for real-time detection	- High training cost in resources - Sensitive to data quality - Overfitting risk
ANN (Artificial Neural Network)	- Accurate classification of malicious requests - Strong with hybrid/statistical integration	- Same general issues as other deep models (complexity, data dependency, overfitting)
NLP + DL (Natural Language Processing with Deep Learning)	- Reduces false positives - Effective in understanding request semantics	- May require tailored preprocessing pipelines - Dependent on high-quality linguistic data
PNN (Probabilistic Neural Network)	- High accuracy (98%) - Performs well with optimization techniques (e.g., BAT algorithm)	- May not generalize well without fine-tuning - Complexity increases with feature space

2.4 Hybrid Algorithms

Hybrid algorithms have risen as a effective approach for identifying SQL injection (SQLI) assaults, combining the qualities of different methods to overcome the restrictions of standalone models. These strategies regularly coordinated machine learning, deep learning, or statistical techniques to improve detection accuracy, strength, and versatility in energetic situations. This segment surveys eminent crossover models utilized within the writing for SQLI location, highlighting the design, preferences, and impediments of each.

1.CNN+BiLSTM:

This hybrid model leverages the feature extraction capabilities of Convolutional Neural Networks (CNN) and the sequential data processing strength of Bidirectional Long Short-Term Memory (Bi-LSTM) networks. CNN is used to extract local features from input queries, while Bi-LSTM captures long-term dependencies in both forward and backward directions (Abebe et al., 2024; Lilhore et al., 2025).

Strengths:

1. Tall precision in identifying complex SQL injection patterns.
2. Compelling highlight extraction and arrangement modeling.
3. Strength against different SQLI assault shapes.

Weaknesses:

1. High computational cost.
2. Requires large datasets for training.
3. Potential risk of overfitting if not regularized properly.

3. FFNN + SVM:

This hybrid model combines a Feedforward Neural Network (FFNN) for beginning highlight change with a Support Vector Machine (SVM) for final

classification. FFNN is capable for capturing non-linear designs, whereas SVM guarantees solid generalization through margin-based classification(Ángeles Rojas et al., 2021; Krishna et al.).

Strengths:

1. Excellent performance in binary classification.
2. Effective in dealing with high-dimensional feature spaces.
3. Reduced overfitting compared to deep networks alone.

Weaknesses:

1. Requires cautious tuning of hyperparameters.
2. FFNN preparing is resource-intensive.
3. SVM can battle with exceptionally huge datasets.

3. CNN + SVM:

This hybrid integrates CNN for extricating significant highlights from input information and SVM for classification. CNN is especially great at recognizing spatial or auxiliary designs, whereas SVM serves as a capable choice boundary(Luo et al., 2019).

Strengths:

1. Strong pattern recognition combined with robust classification.
2. Reduces noise in feature selection.
3. Effective in minimizing false positives.

Weaknesses:

1. Computationally intensive training process.
2. Sensitivity to imbalanced data.
3. Needs extensive preprocessing.

4. LSTM + Random Forest:

In this approach, LSTM is utilized to prepare consecutive input data, whereas Random Forest acts as an gathering classifier that diminishes change and increments expectation solidness. The combination benefits from LSTM's worldly learning and Random Forest's strength (Djaballah et al., 2024).

Strengths:

1. High resilience to overfitting.
2. Handles boisterous data well.
3. Interpretable outfit yield.

Weaknesses:

1. High memory and computation prerequisites.
2. Preparing can be moderate on expansive datasets.
3. Requires successive labeling.

5. ANN + Statistical Models:

This hybrid blends Artificial Neural Networks (ANNs) with factual strategies such as Chi-square or correlation-based sifting. The factual layer handles starting include choice, whereas ANN performs classification(SIAHAAN & GIRSANG, 2023).

Strengths:

1. Improved accuracy through noise reduction.
2. Lightweight compared to deep learning models alone.
3. Adaptable to dynamic SQLI behavior.

Weaknesses:

1. Performance depends on the quality of statistical feature selection.
2. Less effective on unseen attack patterns.

3. May require manual threshold tuning.

To superior get it the capabilities and confinements of different half hybrid algorithms utilized for SQL injection detection, the taking after table presents a disentangled comparison. It highlights the key qualities and shortcomings of each approach, based on later writing. This comparative outline makes a difference distinguish the foremost reasonable models for execution in cybersecurity frameworks depending on asset accessibility, exactness prerequisites, and information complexity.

Table 2.3: Summary and Comparative Table of Hybrid Algorithms

Hybrid Algorithm	Strengths	Weaknesses
CNN + Bi-LSTM	High accuracy (~99.5%), strong in pattern detection, handles sequence data	High computational cost, risk of overfitting, needs large datasets
FFNN + SVM	Balanced classification power, interpretable, adaptable to new patterns	May require feature engineering, sensitive to data noise
CNN + SVM	Strong feature extraction (CNN) and robust classification (SVM)	Computationally intensive, limited flexibility without tuning
LSTM + Random Forest	Sequential modeling (LSTM) + ensemble robustness (RF), improved generalization	Slower training, higher memory use, complexity in integration
ANN + Statistical Features	Accurate with clean features, simple architecture	Limited adaptability, prone to overfitting, needs manual feature tuning

2.5 Comparative Analysis of Algorithms

A comparative analysis of algorithm performance published in the literature is offered in order to comprehend the current state of SQL injection detection techniques. The average accuracy, advantages, and disadvantages of a number of conventional and deep learning methods are compiled in the following table:

Table 2.4 : Comparative Analysis of some Algorithms

Algorithm	**Accuracy (%)**	**Strengths**	**Limitations**
Naïve Bayes	93.3	High accuracy, Simple, Efficient	Assumes feature independence, Struggles with complex data
SVM (Support Vector Machines)	99.0	Handles high-dimensional data, High accuracy	Computationally complex, Sensitive to kernel
Logistic Regression	85.0	Simple, Interpretable	Limited to linear relationships, Sensitive to multicollinearity
Decision Trees	90.0	Simple, Fast training, Visualizable	Prone to overfitting, Instability with small data changes
Passive Aggressive Classifier	79.0	Real-time detection, Efficient updates	Moderate accuracy, Struggles with complexity
Gradient Boosting Classifier	99.3	High accuracy, Flexibility	Computationally expensive, Risk of overfitting
AdaBoost Classifier	99.0	High accuracy, Adaptable, Improves weak classifiers	Sensitive to noisy data, computationally intensive
Convolutional Neural Networks (CNNs)	99.66	High accuracy, Excellent feature extraction	Computational complexity, Data dependency
Recurrent Neural Networks (RNNs) & LSTMs	97.5	Sequential data handling, Flexible	High computational complexity, Overfitting risk
Feed-Forward Neural Networks (FFNNs)	99.0	High accuracy, Robust pattern recognition	Computational complexity, Risk of overfitting
Neural Networks (NNs)	98.0	High accuracy, Adaptable	Computational complexity, Overfitting risk

Artificial Neural Networks (ANNs)	99.0	High reliability, Handles complex patterns	Computationally intensive, Data quality dependent
Natural Language Processing (NLP) with Deep Learning	98.25	Reduces false alarms, Interprets HTTP requests	Complex, Requires significant resources
Probabilistic Neural Networks (PNNs)	98.0	High accuracy, Optimized via BAT algorithm	Limited in handling non-linear data, Demands computational power
CNN + Bi-LSTM	99.4	High accuracy (~99.5%), strong in pattern detection, handles sequence data	High computational cost, risk of overfitting, needs large datasets
FFNN + SVM	89.7	Balanced classification power, interpretable, adaptable to new patterns	May require feature engineering, sensitive to data noise
CNN + SVM	96.2	Strong feature extraction (CNN) and robust classification (SVM)	Computationally intensive, limited flexibility without tuning
LSTM + Random Forest	98.97	Sequential modeling (LSTM) + ensemble robustness (RF), improved generalization	Slower training, higher memory use, complexity in integration
ANN + Statistical Features	89.7	Accurate with clean features, simple architecture	Limited adaptability, prone to overfitting, needs manual feature tuning

This overview provides a foundation for understanding the relative strengths of different methods, particularly in terms of computational cost, adaptability to complex data, and potential limitations under real-world constraints.

2.5.1 Comparison with Literature Results

The classification accuracies attained in this study and those documented in the recent literature for SQL injection detection were thoroughly compared in order to put the research's findings into even more context. Performance across two datasets (Dataset 1 and Dataset 2) is displayed in two tables.

1- Accuracy Comparison for Dataset 1:

Table 2.5 : Accuracy Comparison for Dataset 1

Algorithm	**Count Vectorizer**	**Word Vectorizer**	**TF-IDF**	**Accuracy (Literature)**	**Ref**
RNN	97.85%	98.33%	98.21%	90.24%	(Li et al., 2019)
LSTM	97.5%	98.57%	98.21%	91.53%	(Li et al., 2019)
Bi-LSTM	97.74%	98.33%	98.21%	96%	(Gandhi et al., 2021)
FFNN	97.97%	98.57%	98.21%	98.04%	(Hassan et al., 2021)
NN	97.85%	98.21%	95.71%	96%	(Zhang et al., 2022)
Logistic Regression	92.14%	96.07%	90.95%	92%	(Alghawazi et al., 2022)
Random Forest	89.16%	96.42%	91.19%	93.21%	(Li et al., 2019)
XGBoost	89.17%	95.00%	89.76%	92%	(Gandhi et al., 2021)
Naive Bayes	97.74%	97.98%	97.73%	95%	(Alghawazi et al., 2022)
SVM	91.90%	88.45%	95.6%	90.79%	(Li et al., 2019)
Passive Aggressive	91.90%	96.67%	93.1%	79%	(Alghawazi et al., 2022)
Gradient Boosting	90.11%	95.60%	89.88%	91%	(Gandhi et al., 2021)
AdaBoost	89.52%	95.12%	90.6%	90%	(Gandhi et al., 2021)

Decision Tree	85.48%	89.52%	87.98%	93.57%	(Li et al., 2019)
CNN	97.61%	98.21%	98.21%	98.25 %	(Chen et al., 2021)
FFNN + Naive Bayes	98.1%	89.05%	98.21%	98.7%	)Demilie & Deriba, 2022(
FFNN + SVM	97.74%	91.9%	98.21%	-	-
FFNN + Random Forest	97.62%	93.57%	98.21%	-	-
NN + SVM	98.1%	89.76%	98.21%	98.5%	(Crespo-Martínez et al., 2023)

2- Accuracy Comparison for Dataset 2

Table 2.6 : Accuracy Comparison for Dataset 2

Algorithm	**Count Vectorizer**	**Word Vectorizer**	**TF-IDF**	**Accuracy (Literature)**	**Ref**
RNN	95.15%	98.7%	96.2%	90.24%	(Li et al., 2019)
LSTM	95.1%	98.6%	96.4%	91.53%	(Li et al., 2019)
Bi-LSTM	95.1%	98.8%	96%	96%	(Gandhi et al., 2021)
FFNN	95.25%	98.9%	82.3%	98.04%	(Hassan et al., 2021)
NN	94.85%	98.60%	89.5%	96%	(Zhang et al., 2022)
Logistic Regression	94.05%	97.60%	91.6%	92%	(Alghawazi et al., 2022)
Random Forest	94.95%	75.6%	75.7%	93.21%	(Li et al., 2019)
XGBoost	93.55%	98.60%	98.4%	92%	(Gandhi et al., 2021)

Naive Bayes	77.15%	79.30%	79.2%	95%	(Alghawazi et al., 2022)
SVM	90.08%	88.60%	88.7%	90.79%	(Li et al., 2019)
Passive Aggressive	93.9%	98.50%	93.3%	79%	(Alghawazi et al., 2022)
Gradient Boosting	91.45%	96.10%	97%	91%	(Gandhi et al., 2021)
AdaBoost	93.95%	96.70%	97%	90%	(Gandhi et al., 2021)
Decision Tree	94.8%	75.00%	74.60%	93.57%	(Li et al., 2019)
CNN	95.10%	98.7%	96.4%	98.25 %	(Chen et al., 2021)
FFNN + Naive Bayes	95.55%	85.95%	94.65%	-	-
FFNN + SVM	95.3%	86.4%	94.45%	-	-
FFNN + Random Forest	95.5%	81.8%	94.75%	-	-
NN + SVM	94.9%	85.95%	94.55%	98.5%	(Ángeles Rojas et al., 2021)

2.5.2 Key Observations and Implications

1- Consistent Superiority of Deep Learning Models: Deep learning models such as CNN, FFNN, LSTM, and Bi-LSTM reliably outflanked conventional machine learning calculations over both datasets. For case, the FFNN demonstrate accomplished an accuracy of 98.90% on Dataset 2 when utilizing Word Vectorizer, which somewhat outperforms already detailed comes about within the writing (Hassan et al., 2021). This steady execution highlights the viability of profound models in capturing complex designs in literary information.

2- Sensitivity of Naïve Bayes to Dataset Complexity: Naïve Bayes appeared striking inconstancy over datasets. Whereas it accomplished tall accuracy on Dataset 1 (97.98% with Word Vectorizer), its execution dropped essentially on Dataset 2 (79.30%). This decay is particularly apparent when utilizing Word Vectorizer, underscoring Naïve Bayes' known impediments in taking care of complex, high-dimensional feature spaces.

3- Robustness of Logistic Regression: Logistic Regression demonstrated steady and competitive execution over both datasets, achieving up to 97.60% accuracy on Dataset 2 with Word Vectorizer. These come about and adjust well with values detailed in past ponders(Alghawazi et al., 2022), demonstrating the model's unwavering quality in different literary classification scenarios.

4-Effectiveness of Word Vectorizer for Deep Models: Word Vectorizer by and large driven to made strides accuracy, especially for deep learning models. For occasion, the Bi-LSTM show progressed from 95.10% (with Count Vectorizer) to 98.80% (with Word Vectorizer) on Dataset 2. This recommends that dispersed representations capture semantic connections more successfully than scanty representations like Count Vectorizer or TF-IDF.

5-Limitations of Tree-Based Models with Word Embeddings: Traditional tree-based models such as Random Forest and Decision Tree displayed debased execution when combined with Word Vectorizer, particularly on Dataset 2. Random Forest, for example, dropped from 94.95% (Count Vectorizer) to 75.60% (Word Vectorizer). This decrease is likely due to the trouble these models confront when preparing thick, continuous vector embeddings, which may not adjust well with their structure-based learning instruments(Li et al., 2019).

2.6 Hyperparameter Importance and Impact on Model Performance

The algorithms utilized to distinguish SQL injection attacks, besides the components and characteristics that contribute to their exactness and optimization, have been the subject of various ponderings. A number of hyperparameters, including group estimate, number of ages, optimizer, arbitrary state, and test measure, must be chosen

carefully in order to ensure the rigor and repeatability of demonstration preparation. The model's generalization capacity, preparing stability, and performance are all significantly affected by these components.

A few studies have illustrated that the precision and execution length of the demonstration are specifically affected by these hyperparameters. A careful depiction of each hyperparameter, its work, and how it applies to different model types can be found underneath:

1. Batch Size:

1. Applicable to: Recurrent neural networks (RNNs), feedforward neural networks (FFNNs), long short-term memory (LSTM), bidirectional LSTM (Bi-LSTM), convolutional neural networks (CNNs), and general neural networks (NNs) are examples of deep learning models that can be used with this technique.
2. Effect: The batch size determines how many training samples are processed prior to updating the internal parameters of the model. Memory use, training speed, and the generalization capacity of the model are all directly impacted. A popular batch size that strikes a compromise between model performance and computational economy is 32(Kandel & Castelli, 2020).
   Larger batch sizes can speed up training by utilizing hardware parallelism but may lead to overfitting and reduced generalization.
   Smaller batch sizes improve generalization but may result in noisier gradient estimates and longer training times (Kandel & Castelli, 2020).
3. Not Applicable to: Conventional machine learning models (e.g., Random Forests, Decision Trees, Gradient Boosting, Naïve Bayes, Logistic Regression, Passive Aggressive Classifier, SVM) either do not require explicit batch-based training or use the complete dataset at once.

2. Epochs:

1. Applicable to: All neural network-based models (CNNs, RNNs, LSTM, Bi-LSTM, FFNN, NNs).

2. Effect: How many times the complete training dataset is run through the model is determined by the number of epochs. The model can usually learn more intricate patterns with more epochs, which increases accuracy. The usage of 50 epochs is a standard procedure (Goodfellow, 2016).

   Overfitting, however, can result from using too many epochs, especially on small or noisy datasets. Training efficiency and generalization are enhanced by strategies like early stopping, which help end training when validation performance plateaus or decreases (Li et al., 2020).
3. Minimal Impact on: Traditional models (Logistic Regression, SVM, Naïve Bayes, Decision Trees, AdaBoost, Gradient Boosting) where training is performed through direct optimization or fixed rounds rather than multiple epochs.

3. Optimizer:

1. Applicable to: As it were, neural network-based models (CNNs, RNNs, LSTM, Bi-LSTM, FFNN, NNs).
2. Effect: Optimizers control how the model's weights are updated based on the computed angles. The choice of optimizer essentially impacts meeting speed, preparing stability, and by and large performance. The Adam optimizer is broadly utilized in deep learning for its versatile learning rate instrument, empowering speedier learning compared to conventional optimizers like SGD(Chollet, 2021; Kingma, 2014). Later variations like AdamW make strides in regularization, particularly for large-scale neural systems(Reyad et al., 2023).
3. Not Used in: Traditional machine learning calculations (e.g., SVM, Decision Trees, Naïve Bayes, Logistic Regression, AdaBoost, Gradient Boosting), which utilize deterministic or ensemble-based optimization strategies without requiring angle plunge.

4. Random State:

1. Applicable to: All models, including any shape of randomness—data part, demonstrate initialization or stochastic computations (e.g., Decision Trees, Random Forests, Neural Networks).

2. Effect: Setting a settled random state guarantees reproducibility. It controls random processes like dataset parts and parameter initialization. This consistency is pivotal in test investigations to ensure that comes about can be reproduced(Chollet, 2021).In models such as Choice Trees and Irregular Woodlands, arbitrariness influences highlight determination and parts. A settled arbitrary state makes a difference in guaranteeing steady behavior over different runs. It is additionally vital in neural systems, where irregular initialization can affect learning and execution(Gallicchio & Scardapane, 2020).
3. Not Applicable to: Algorithms like Naïve Bayes or Logistic Regression don't depend on irregular initialization or stochastic operations during preparation.

5. Test Size:

1. Applicable to: All machine learning and deep learning models.
2. Effect: The percentage of the dataset put aside for model evaluation depends on the test size. A typical allocation is 20% for testing and 80% for instruction. This guarantees that the model is fairly assessed on unseen data and has sufficient data to learn from.

   A straightforward train/test split could result in significant variance for small datasets. By evaluating the model on several non-overlapping subsets, k-fold cross-validation is recommended in these situations to guarantee more accurate performance estimations (Liengaard et al., 2021; Phinzi et al., 2021). This is particularly helpful for models that are sensitive to changes in the dataset, including ensemble techniques like AdaBoost and Gradient Boosting.
3. Not Applicable to: None. Data splitting is necessary for all models, while the exact approach relies on the size of the dataset and the model's susceptibility to overfitting.

2.7 Research Gaps in Current Studies

Indeed, in spite of the fact that SQLI location has advanced, there are still a few confinements to address:

1- Advanced Attack Techniques: One of the primary challenges for current models is detecting newer and more complex SQLI techniques, such as polymorphic attacks.
2- Real-Time Detection: Most existing strategies are, as it were, effective in scenarios requiring offline investigation, as they are not optimized for real-time discovery.
3- Diversity of Datasets: Many studies use artificial datasets, which might not accurately represent assault patterns in the actual world. This restricts these models' capacity to be used in a variety of real-world situations.

3. Chapter Three: Background and methodology

3.1 SQLI Attack Overview

In addition to providing an overview of injection attacks, this section of the article also covers their targets, origins, and various varieties. Table 1 provides a final summary of all of this.

3.1.1 SQLI Attack Sources

Any application that makes use of a database that is either hidden or improperly exposed is vulnerable to injection attacks and other threats, thus it needs to be adequately secured. User input, server variables, cookies, and stored injections are the four primary sources that any attacker frequently targets (Deriba et al., 2022).

1. User Input Injection: In order to attack web forms, such as those used to log into a specific page or website, hackers usually target passwords, banking information, personal information, etc. This is achieved by secretly introducing malicious code.
   For Example: In the username field, the attacker inputs:
   ' OR '1'='1
   And leaves the password field empty.
   The resulting SQL query might look like this:
   SELECT * FROM users WHERE username = '' OR '1'='1' AND password = '';
   Since '1'='1' is always true, the query returns all users, granting unauthorized access to the system.
2. Cookie-Based Injection: Cookies, which are used to store user preferences and session information, are the target of several attacks. in order to exploit online apps that depend on these cookies and to see if malware has been installed and saved on the devices being utilized. obtaining unauthorized access or maybe taking over user sessions.
   For Example: Consider a website that uses a cookie named session_id to track user sessions. An attacker modifies the cookie value to:
   1234'; DROP TABLE users; --

The dangerous DROP TABLE command may be executed in the ensuing SQL query, erasing the users table, if the server processes this cookie without validation.

3. Server Variable Injection: The goal of this attack is to infiltrate server factors, such as HTTP headers, or organize distinctive data. If they are not thoroughly examined, attackers may introduce dangerous code while posing as real security threats.

   For Example: An attacker crafts a malicious User-Agent header as follows:
   Mozilla/5.0' UNION SELECT credit_card_number FROM customers; --
   If this header is directly used in an SQL query, it could leak sensitive data.

4. Stored Injection: This technique, sometimes referred to as second order infusion, involves inserting malicious inputs into the database since each time these inputs are used Enacted SQLI An attacker can obtain information without permission, for instance, by utilizing a phone username. This is an illustration:

```
queryString=" UPDATE employees SET
password='"
+ newPassword +"' WHERE
employeeName='" +
employeeName +"' AND password='"
+oldPassword
+ """
```

3.1.2 SQLI Attack Goals

The hacker or attacker has multiple goals in mind when carrying out SQL injection attacks. These objectives are frequently driven by elements like monetary gain, data theft, sabotage, or competitive advantage. Below is a summary of these objectives and examples supporting each one:

1. Identifying Injection Points: Aggressors start by recognizing the factors they will abuse such as shape areas treats or other inputs These factors permit the infusion of malevolent code for occasion a programmer might test a login shape by entering SQL commands to see how the application reacts hence revealing vulnerabilities in its SQL preparing.

2. Database Fingerprinting: To create effective attack plans, the attacker selects the database sort and adaptation. This information enables them to modify SQL queries to make use of specific features. For example, if the attacker discovers that the database is MySQL version 8.0, they may design unique payloads that take advantage of version-specific flaws.
3. Mapping the Database Structure: Attackers must become familiar with the database pattern in order to craft precise queries for information extraction or manipulation. By crafting error messages, they may find table names such as clients or column names such as password hash, which might then be exploited.
4. Data extraction: Numerous SQL combination ambushes indicate the recovery of sensitive data as well as the verification of personal information, budgetary records, and login credentials. A UNION SELECT request for an event may well be utilized by an assailant to blend their harmful request with significant data recuperation shapes.
5. Modifying Database Content: In order to obtain additional power, an attacker or hacker may add or incorporate malicious links or alter or remove items from the database. They might, for instance, alter a product's description on an online store to contain a phishing link in case the user clicks on it.
6. Starting a Denial of Service (DoS) attack: Tables may be locked or deleted in this kind of denial-of-service attack, rendering the database unavailable to authorized users. For instance, an attacker runs a command that overloads the database with resource-demanding queries, resulting in the database crashing or becoming overloaded.
7. Avoiding Authentication: An attacker can escalate privileges to access restricted regions or obtain unauthorized access by evading authentication protocols. For instance, they might be able to compromise security by logging in as an administrator without the required credentials if they type OR '1'='1 in the login field (Halfond et al., 2006).
8. Executing Remote Commands: An attacker can escalate privileges to access restricted regions or obtain unauthorized access by evading authentication protocols. For instance, they might be able to compromise security by logging in as an administrator without the required credentials if they type OR '1'='1 in the login field (Halfond et al., 2006).

9. Privilege escalation: An attacker can increase their privileges in a specific system they can access by taking advantage of flaws. To obtain complete or partial control over a database, they can, for instance, alter user roles to grant themselves administrator access.

3.1.3 Types Of SQLI

In order to choose unauthorized data, delete data, or run malicious code, each type of SQL injection attack uses a unique input that addresses weaknesses. A comparison flowchart, relief techniques, and a graph of the top categories may be found below:

1. Tautologies: Execution flaws are abused by the attacker to increase their advantages and gain access to all information, potentially changing it. Such an attack is deemed extremely dangerous due to its crucial impact.
   Mitigation: Implement strict input validation and use parameterized queries.
2. Blind SQL Injection: Execution flaws are abused by the attacker to increase their advantages and gain access to all information, potentially changing it. Such an attack is deemed extremely dangerous due to its crucial impact(Demilie & Deriba, 2022).
   Example: Using AND 1=1 (true) or AND 1=2 (false) to observe behavior changes.
   Mitigation: Disable detailed error messages and use prepared statements.

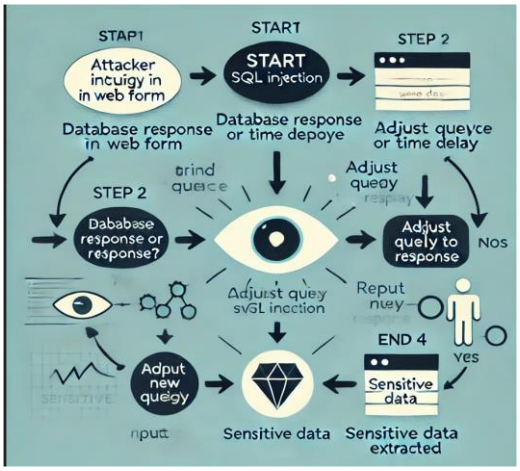


Figure 3.1 : Blind SQL Injection

3. Union Query: Aggressors utilize the UNION administrator to combine a pernicious enquiry with the initial enquiry, permitting them to recover information from different database tables.

   Example: Adding UNION SELECT visacardNo FROM CreditCards WHERE acctNo=10032 -- can reveal sensitive information like credit card numbers. Mitigation: Restrict permissions for SQL accounts and validate input length and structure.
4. Piggy-backed Query: Attackers append additional malicious queries to the original query using the query delimiter, enabling them to extract, edit, or add data.

   Example: Adding; DROP TABLE userDetails; to a valid query could delete a critical table.

   Mitigation: Use query parameterization and avoid dynamic SQL construction.
5. Stored Procedures: Attackers can use stored procedures that communicate with the operating system to carry out malicious commands to get access. Examples include the alteration of files or the installation of malware when malicious input is injected into a procedure that communicates with the file system. Attackers can use stored procedures that communicate with the operating system to carry out malicious commands to get access. Examples include the alteration of files or the installation of malware when malicious input is injected into a procedure that communicates with the file system.

   Mitigation: Validate inputs within stored procedures and restrict their privileges to minimize potential risks.
6. Alternate Encoding: Attackers alter the encoding of injected text to bypass detection mechanisms.

   Example: Using char(0x73687574646f776e) (which translates to "SHUTDOWN") can trigger a database shutdown.

   Mitigation: Normalize input data before validation and monitor for unusual query patterns.
7. Illegal/Logically Incorrect Queries: Attackers can reveal information about the database schema by submitting queries that are erroneous, which will result in error messages. They can then hone their next attacks using this knowledge(Chen et al., 2021).

8. Example: A malformed query may expose table or column names in the error message.
   Mitigation: Suppress detailed error messages and sanitize all inputs. Comparison Table

SQL injection attacks take advantage of flaws in input preparation to accomplish various malicious objectives, like gaining unauthorized access to data or executing code. Every type of SQL injection has unique properties. levels of complexity and methods of relief an overview of the various assault types is provided in the table below, emphasizing their complexity, prevalence, and possible impact:

Table 3.1 :Comparison Table

Attack Type	**Complexity**	**Prevalence**	**Impact**
Tautologies	Low	High	High risk of privilege escalation
Blind SQL Injection	Medium	High	Data leakage
Union Query	Medium	High	Data extraction
Piggy-backed Query	High	Medium	Data modification/deletion
Stored Procedures	High	Low	Malware execution
Alternate Encoding	Medium	Medium	Evasion of detection mechanisms
Illegal/Logically Incorrect Queries	Low	Medium	Information disclosure

To obtain unauthorized access, harvest data, or run malicious code, these attack types take use of various input processing flaws (Chen et al., 2021; Demilie & Deriba, 2022). Included is a table that lists all SQLI attack types, sources, and goal classifications.

TABLE 3.2: SQLI Attack Sources, Types and Goals classification:

Table 3.2 : Source, Goals and Types Attack

Attack Sources	**Attack Goals**	**Attack Types**
-User input -Cookies -Server variables - Stored Injection	- Identifying Injection Points - Database Fingerprinting - Mapping the Database Structure - Data extraction	-Tautology -Illegal/logically - incorrect queries -Union query

	- Starting a Denial of Service (DoS) attack - Modifying Database Content - Avoiding Authentication - Executing Remote Commands - Privilege escalation	-Piggyback query -Stored procedure -Inference -Alternate encoding

3.2 Characterization of SQL Injection Detection

Using data divided into two primary categories—explanations of SQL infusion and non-SQL infusion statements that identify SQL infusion— In general, SQLI is a classification problem. Non-SQL infusion explanations include normal content and standard SQL explanations. Within the dataset, SQL infusion articulations are designated as 1 and non-SQL infusion articulations are identified as. The goal is to identify malicious SQL injection statements and distinguish them from harmless ones, such text and standard SQL.

3.2.1 SQL Injection Statements

SQL injection statements typically contain specific keywords like "*", ";", "_", "-", "(", ")", "where", "select", "update", "insert", and "drop". Based on the dataset, SQL injection statements are classified into several types:

1. Repetition Attack: circumvents authentication and gains access to private information by using true-true conditions in query clauses.
2. Illegal Comment Attack: stops valid SQL code from running by inserting comment characters (such as "--", "#", "/*") inside SQL statements.
3. Union Query Attack: Uses the "union" keyword to append malicious SQL code, extracting additional information by manipulating query results. Example: SELECT * FROM users UNION SELECT username, password FROM admin;

4. Explicit Error Injection: reveals injectable parameters and database types by executing illicit SQL queries to force error pages. Example: Table names may be revealed in error messages returned by a malicious query.
5. Boolean Blind Injection: submits queries that make sense (AND 1=1 vs. AND 1=2) and watches for injection by comparing the answers.
6. Time Blind Injection: Inserts time delay functions (e.g., WAITFOR DELAY '00:00:05') to detect injection by observing response delays.
7. Mult statement Attack: Uses query separators (e.g., ;) to add additional queries, modifying or extracting data or executing commands. Example: SELECT * FROM users; DROP TABLE users;

These attacks misuse specific catchphrases and techniques to get beyond database security, allowing attackers to carry out illegal actions. Some more recent techniques, such as the use of unique encodings, can make the attack more difficult to detect, making it more difficult for security systems to identify and stop these infusions(Demilie & Deriba, 2022).

3.2.2 Non-SQL Injection Statements

1. Plain Text Statements: These consist of standard characters, numerals, and letters. They show no indications of malevolent intent and are easily recognized as safe. A false positive could occur, though, if a legitimate SQL statement occasionally appears as a straightforward text statement. For instance, detection algorithms may wrongly label a user's inclusion of a comment or description in a query as suspicious. Algorithms can more effectively distinguish between these situations by examining the query's intent as well as the larger context, which includes the user's previous behavior patterns and actions.

2. General SQL Statements: Database scheduling uses these terms, which are also used in innocuous, everyday requests, therefore even if they are utilized in injection attacks, they are not regarded as malicious. Depending on who is using it and the situation, the query's content can either be harmful or beneficial. For detection systems to prevent false

positives and guarantee correct categorization, certain contextual aspects must be taken into consideration.

3.False Positives and Negatives: Algorithms must balance sensitivity when distinguishing between SQL and non-SQL statements in order to prevent these kinds of errors. False positives could lead to needless investigations when valid inquiries are mistakenly categorized as threats. However, assaults may go undetected due to false negatives, which occur when malicious queries are not found. To minimize results and guarantee that queries are categorized based on their actual context and intent, these systems must be set up correctly (Jothi et al., 2021).

3.2.3 Data Characterization

Words like "select," "*," and "from" are commonly used in both SQL injection and non-SQL injection commands. It is challenging to use these terms as reliable classification indicators because of their overlap. Take a look at the accompanying table and word cloud to better understand this challenge:

Table 3.3 : Commonly Occurring Terms in SQL and SQL Injection Statements

Term	Occurrence in SQL Commands	Occurrence in SQL Injection
SELECT	Common	Common
*	Common	Common
FROM	Common	Common
WHERE	Common	Occasionally Present
UNION	Rare	Common (in Union Injection)
--	Rare	Common (in Comment Injection)

This table highlights that many keywords, such as "SELECT","*," and "FROM," frequently appear in both SQL commands and SQL injection attacks, complicating the task of distinguishing between the two. Certain terms, like "UNION" and "--," are more commonly associated with SQL injection attacks, making them more valuable for identifying malicious queries.

-Word Cloud Visualization:

The distribution of terms is shown visually in a word cloud next to the table. It would make words like "SELECT," "FROM," and "*" look larger because they are used frequently in both beneficial and detrimental searches. The overlap and difficulties in telling the two apart are highlighted by this graphic (Ghosh et al., 2024; Sun et al., 2023).
-Role of Feature Engineering:

Addressing these overlaps requires the use of feature engineering. We can increase classification accuracy by developing more sophisticated features that take into account not just the frequency of keywords but also their context, including surrounding phrases, query structure, and the kind of operation being carried out. While SELECT is frequently used in both kinds of queries, for instance, a query that has "SELECT" followed by "UNION" or "--" can be regarded as suspicious. This procedure is improved by advanced feature construction, which facilitates the detection of possible threats (Jothi et al., 2021; Kakisim, 2024).

3.3 Count Vectorization vs Word Vectorization vs TD-IDF

1-Count Vectorization is a common method for turning text into numerical representation in natural language processing (NLP). The basic way it works is by entering the frequency of each word's occurrences in a document into a matrix. Every push in this lattice denotes a different report, and every column refers to a dictionary phrase. Each term's recurrence inside the related content appears within the framework's cells. Actually, this method works well for a lot of content categorization problems and can yield high-dimensional data, especially when dealing with large vocabulary sets (Jiao & Zhang, 2021).

Let's illustrate this with an example. Think of a table that shows the word counts for each document, where the frequency of each phrase in a given text is indicated by a cell.

-Visual Example: Comparing Count Vectorization and Word Vectorization Let’s use a simple SQL query as an example:

SELECT * FROM users WHERE name = 'John';

-Count Vectorization: In this case, a matrix is generated where each word from the vocabulary (e.g., "SELECT", "FROM", "users", "WHERE", "name", "John") is assigned a count. The output is a vector where the count for each word appears in its respective position (Bommasani et al., 2020).

The picture underneath shows a portion of the application in action. It changes over content information into vectors utilizing Count Vectorizer, at that point parts them into training and test sets to prepare them for machine learning models.

```
# Vectorize the sentences
vectorizer = CountVectorizer(min_df=2, max_df=0.9, stop_words='english')
posts = vectorizer.fit_transform(df['Sentence'].values.astype('U')).toarray()
```

Figure 3.2: Count vectorization code

2-Word Vectorization (e.g., Word2Vec, GloVe): This method uses a high-dimensional space with continuous vectors for each word. However, this method takes into account a word's context and deeper semantic significance in addition to its frequency of occurrence. The name "John" would have a totally different vector that represents its distinct identity and function inside the sentence, whereas terms like "SELECT" and "FROM" might have comparable vectors since they have similar grammatical tasks (Kenton & Toutanova, 2019).

This code, Word2Vec, is used to train a model that creates vector representations (embeddings) of words according to their context inside sentences, as seen in the figure below. This code initializes an embedding matrix with zeros and generates a vocabulary. The appropriate vector, if any, from the Word2Vec model is then assigned to each word in the lexicon.

```
word2vec_model = Word2Vec(sentences=[seq.split() for seq in sentences], vector_size=EMBEDDING_DIM, window=5, min_count=1, workers=4)
vocabulary_size = len(word_index) + 1
embedding_matrix = np.zeros((vocabulary_size, EMBEDDING_DIM))
for word, i in word_index.items():
    if i < vocabulary_size and word in word2vec_model.wv:
        embedding_matrix[i] = word2vec_model.wv[word]
```

Figure 3.3: Word vectorization

3. TF-IDF (Term Frequency - Inverse Document Frequency)

TF-IDF is an improvement above basic count-based techniques since it modifies word frequency based on how unique or informative a word is across several pages. It strikes a balance between two metrics:

A word's term frequency (TF) indicates how frequently it occurs in a document. By determining how uncommon a word is in all papers, Inverse Document Frequency (IDF) lessens the weight of popular words. The formula for TF-IDF is:

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log(\text{DF}(t)N)$$

Where:

- t is the term,
- d is the document,
- N is the total number of documents,
- DF(t) is the number of documents containing term ttt.

TF-IDF emphasizes terms that are important within a specific document but not frequent across the whole corpus, making it highly effective in distinguishing keywords and reducing the impact of stop words (e.g., "the", "is", "from") (AlShammari; Rajagukguk et al., 2024).

The figure below shows how text data is transformed into TF-IDF features using sklearn's TfidfVectorizer function, which yields a dense matrix. Using the inverse document frequency (TF-IDF) measure, TfidfVectorizer assists in converting text data into a numerical format.

```
# Vectorize text data using TF-IDF
def wordVectorizer(text_data):
    vectorizer = TfidfVectorizer()
    word_vectors = vectorizer.fit_transform(text_data)
    return word_vectors.toarray()
```

Figure 3.4:TF-IDF Vectorization

Use Case Example:

In SQL logs, terms like "SELECT" or "WHERE" may show up as often as possible over numerous inquiries, so their IDF values would be low. Be that as it may, an uncommon table title like "user_activity_logs" would have the next weight, highlighting its importance in particular settings.
TF-IDF is especially useful for:

- Spam detection
- Document classification
- Information retrieval
- Keyword extraction

While TF-IDF does not capture word semantics like Word2Vec, it is often simpler to compute and performs well for traditional ML models (e.g., Naïve Bayes, SVM) (Grootendorst, 2022).

3-Dimensionality Reduction Techniques

One of the main issues with check vectorization is that it frequently occurs in extremely tall dimensional data, which can be difficult to monitor, particularly when managing large datasets or extensive vocabularies. Reduction techniques like t-SNE and First Component Examination PCA are frequently used to address this dimensionality.

By predicting the information and reorganizing it onto more logical components, PCA makes a difference. This allows us to store the majority of the basic data while also making it easier to deal with. That being said, t-SNE surpasses expectations when it comes to visualizing high-dimensional information. It reorganizes the distinctive evidence of designs and provides experiences by projecting intricate data into two or three dimensions (Brown et al., 2020; Mars, 2022).

-Use Cases Where Word Vectorization and TF-IDF Outperform Count Vectorization

Use Cases Where Word Vectorization and TF-IDF Perform Better Than Count Vectorization In situations that call for a more thorough comprehension of word meaning and context, word vectorization and TF-IDF typically outperform simple count

vectorization. Count vectorization works well for simple tasks like categorizing articles based only on term frequency, but it frequently fails in more complex applications that need semantic comprehension or weighting terms according to their significance within a corpus (Sun et al., 2023).

Examples Where Word Vectorization and TF-IDF Provide Superior Performance:

1-Content Similarity:

Word embeddings (such as Word2Vec or GloVe) capture semantic relationships between words in a continuous vector space, enabling effective measurement of how similar different queries or documents are to each other. Similarly, TF-IDF enhances document comparison by down-weighting frequently occurring terms (e.g., “SELECT”, “FROM”) and emphasizing more informative terms unique to specific documents.

2-Sentiment Analysis:

Word vectorization exceeds expectations at distinguishing assumption by considering the setting in which words show up. It can recognize between comparative expressions with diverse passionate tones. On the other hand, TF-IDF contributes by emphasizing words that are nostalgically critical but happen occasionally over the dataset, such as “terrible” or “excellent.” (AlShammari; Arsyah et al., 2024; Grootendorst, 2022).

3-Keyword Extraction and Topic Modeling:

TF-IDF is particularly compelling in extricating significant catchphrases and expressions that offer assistance in understanding the most points of reports. It plays a central part in models like BER Topic, where TF-IDF is utilized to weight terms inside found themes, making theme names more interpretable.

4-Advanced NLP Tasks:

For more complex applications like machine interpretation, address answering, or summarization—where understanding the complete meaning and connections between words is essential—word embeddings offer rich, contextualized representations. Whereas TF-IDF does not capture setting within the same way, it remains a solid pattern for document-level highlights in content classification and clustering errands.

5-Visual Comparison of Techniques:

1. Count Vectorization: Employing an inadequate document-term framework where each word is represented based on its crude recurrence. Needs semantic understanding.

2. TF-IDF: Upgrades frequency-based representation by weighting words based on their significance over records, decreasing the effect of common but uninformative words.
3. Word Vectorization: Speaks to words as thick vectors that encode semantic meaning and relevant connections based on their utilization in expansive corpora.

-Dimensionality Reduction: High-dimensional vectors from check and TF-IDF strategies can be streamlined utilizing PCA or t-SNE to imagine or make strides preparing proficiency.

Table 3.4 : Comparison Between Count Vectorization, TF-IDF, and Word Vectorization

Feature/Technique	**Count Vectorization**	**TF-IDF**	**Word Vectorization**
Representation	Sparse matrix of word counts	Weighted matrix based on frequency and document rarity	Dense vectors learned from large corpus (e.g., Word2Vec)
Semantic Understanding	None	Limited (based on term importance)	High (captures word meaning and context)
Context Awareness	No	No	Yes
Handling Synonyms	Poor	Better than count, but still limited	Good (similar meanings mapped closely in vector space)
Use Cases	Simple text classification	Keyword extraction, topic modeling	Sentiment analysis, semantic similarity, translation
Dimensionality	High (sparse)	High (sparse)	Lower (dense)

Interpretability	High	Medium	Low (requires decoding vector meaning)

Summary:

Count Vectorization is still helpful for easier classification issues, even if TF-IDF and word vectorization techniques like Word2Vec perform better in jobs requiring a deeper comprehension of language, like sentiment analysis, keyword extraction, and topic modeling. Although more recent developments in natural language processing, like contextual embeddings like BERT and ELMo, provide strong capabilities for capturing deep semantic meanings, they can be restrictive in research settings with limited resources due to their high computational requirements and large training data sets.

Considering these practical limitations—including the need for interpretability, computational efficiency, and compatibility across various traditional and deep learning algorithms—Count Vectorizer, TF-IDF, and Word2Vec were chosen as the most balanced and effective feature extraction techniques for this study. These methods strike a practical balance between simplicity, performance, and contextual awareness, making them well-suited for the objectives and scope of this research (Kenton & Toutanova, 2019; McCormick, 2016; Mikolov et al., 2013; Pennington et al., 2014).

3.4 Assessing Machine Learning Models: Recall, Accuracy, F1 Score and Precision

Accuracy, recall, and precision are crucial measures in machine learning that are used to assess a model's performance, especially when it comes to categorization issues.

1. Accuracy

The proportion of accurately predicted cases either true positives or true negatives to the total number of cases is known as precision.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

In terms of the confusion matrix, accuracy can be expressed as:

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

where:

- TP (True Positives): The model correctly predicts the positive class.
- TN (True Negatives): The model correctly predicts the negative class.
- FP (False Positives): The model incorrectly predicts the positive class.
- FN (False Negatives): The model incorrectly predicts the negative class.

2. Precision

The ratio of accurately anticipated positive instances to all expected positive instances is known as precision. It indicates the proportion of expected positive cases that turned out to be positive.

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

High precision indicates a low false positive rate.

3. Recall (Sensitivity or True Positive Rate)

The percentage of accurately predicted positive examples to all instances that genuinely fall into the positive class is known as recall. It assesses how well the model can account for every positive example.

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

High recall indicates that the model captures most of the actual positive instances.

4. F1 Score

The F1 score, which is the harmonic mean of precision and recall, is a straightforward statistic that strikes a balance between the two. It is useful in situations when you need to balance recall and precision, especially when your class distribution is irregular.

$$\text{F1 Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The compromises of the measures More false positives are frequently the price paid for having a high recall. The user experience may be hampered, or needless alarms may arise. Focusing on high precision, on the other hand, increases the likelihood of missing some positive cases and decreases recall by decreasing false positives.

-How to Handle Unbalanced Datasets?

Applications pertaining to security frequently have unbalanced datasets. For instance, the positive class (such SQL injection attempts) is usually much less common than the negative class (normal traffic). This disparity could skew the assessment metrics. To ensure more balanced results, this can be addressed by using techniques like class weighting or oversampling the minority class.

For example, this figure example to confusion matrix to feedforward Neural Network:

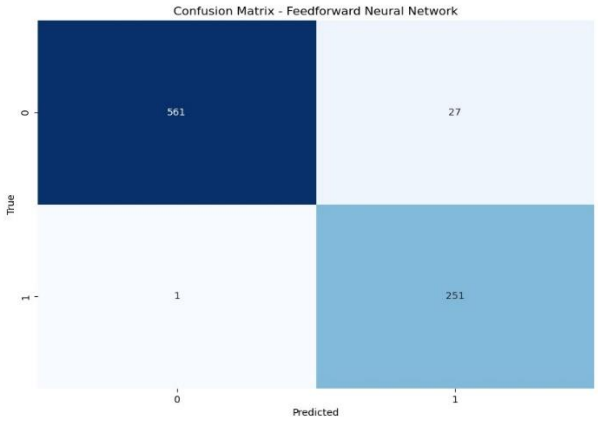


Figure 3.5 : Confusion Matrix of FFNN

TP = 561

FP = 27

FN = 1

TN = 251

1- Precision

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

$$= 561/(561 + 27)$$

=0.954

2- Recall

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

$$= 561/(561 + 1) = 0.998$$

3- F1 Score

$$\text{F1 Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

$$= 2 * \frac{0.954 * 0.998}{0.954 + 0.998}$$

$$= 0.975$$

4- Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FB + FN}$$

$$= \frac{561 + 251}{561 + 251 + 27 + 1}$$

$$= 0.966$$

And another example, this figure example to confusion matrix to SVM in dataset1 use word vectorization:

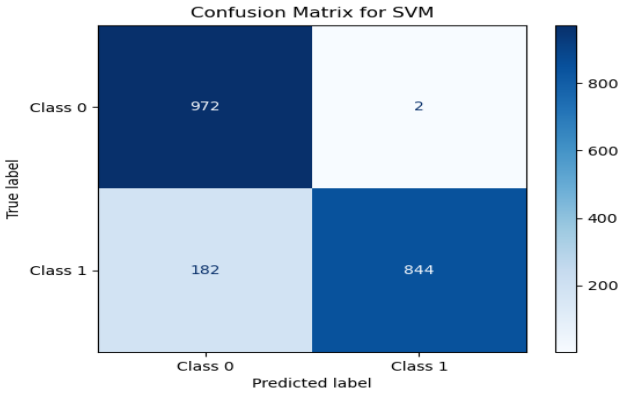


Figure 3.6 : Confusion Matrix of SVM

TP = 972

FP = 2

FN = 182

TN = 844

1-Precision

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

$$= 972/(972 + 2)$$

$$=0.997$$

2-Recall

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

$$= 972/(972 + 182)$$

$$= 0.842$$

3-F1 Score

$$\text{F1 Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

$$= 2 * \frac{0.997 * 0.842}{0.997 + 0.842}$$

$$= 0.913$$

4-Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FB + FN}$$

$$= \frac{972 + 844}{972 + 2 + 182 + 844}$$

$$= 0.908$$

3.5 Methodology

A comparative analysis of unmistakable algorithms for identifying SQL infusion in web applications is the main focus of this thought piece. In order to pick the first feasible calculation in terms of accuracy and execution time, the technique consists of gathering datasets, eliminating highlights using two specific material vector procedures, and choosing and surveying computations. The following are the specific steps:

1.Data Collection

This study used two different datasets to ensure the generalizability and accuracy of the results. Protego and Kaggle, two publicly available sources, provided these datasets, referred to as Datasets 1 and 2, respectively. Because both datasets contain both malicious and legitimate SQL queries, they can be used to test and train machine learning algorithms. The image below shows how the datasets are called.

```
# Load dataset
df = pd.read_csv('Dataset1.csv', encoding='utf-16')
```

Figure 3.7 :Load Dataset

2. Feature Extraction

To prepare text data for classification, I implemented three separate codes, each using a different vector technique:

1. Count Vectors: In the first code, this technique transforms SQL queries into numerical feature vectors by counting the frequency of words in the text.
2. Word Vectors: In the second code, this technique groups words into dense, fixed-size vectors based on their semantic meaning, enabling models to more effectively capture word relationships.
3. TF-IDF Vectors: In the third code, the TF-IDF (Term Frequency - Inverse Document Frequency) technique is used. This technique transforms text data by considering the frequency of words in each document (term frequency) and the importance of those words across the entire text (inverse document frequency). This technique helps highlight the most unique and important words in the context of SQL queries, improving classification performance in tasks such as SQL injection detection.

   Here's an image of a piece of code using TF-IDF vectors:

```
# TF-IDF Vectorization
vectorizer = TfidfVectorizer(min_df=2, max_df=0.9, stop_words='english')
X = vectorizer.fit_transform(df['Sentence'].values.astype('U')).toarray()
y = df['Label']
```

Figure 3.8:TF-IDF Vector

3. Algorithm Selection:

Chosen a range of deep learning and conventional methods for testing. The algorithms were chosen due to their widespread use and demonstrated ability to recognize patterns and features in both structured and vector data. The following are the chosen algorithms:

Table 3.5 : Algorithms Selected

Traditional Algorithms	**Deep Learning Algorithms**	**Hybrid Algorithms**
Naïve Bayes (NB)	Convolutional Neural Networks (CNNs)	FFNN + Random Forest (RF)
SVM (Support Vector Machines)	Recurrent Neural Networks (RNNs)	FFNN + SVM
Logistic Regression	Long Short-Term Memory (LSTM)	FFNN + Naïve Bayes (NB)
Decision Trees	Feed-Forward Neural Networks (FFNNs)	Neural Networks (NNs)+ SVM
Passive Aggressive Classifier	Neural Networks (NNs)	-
Gradient Boosting Classifier	bidirectional long short-term memory (Bi-LSTM)	-
AdaBoost Classifier	-	-
Random Forest (RF)	-	-
XGBoost	-	-

4.Objective:

There are several objectives in the research, but the most important is to identify the algorithm that achieves the highest accuracy in the least time and is excellently efficient. And adaptability to variables such as changing datasets and other methods.

5.Experimental setup:

1. Data Preprocessing: To ensure the validity and effectiveness of the test, both datasets were preprocessed, and then I used the above two different techniques to

transform the input text into vectors, and missing values were handled accordingly.

2. Training and Test Split: To evaluate the models' ability to generalize, each dataset was divided into training and test sets in an 80:20 ratio. Practical experiments can be used to modify the ratio.
3. Hyperparameter Tuning: The hyperparameters of each algorithm were tuned using techniques such as grid search and cross-validation to achieve optimal performance. A portion of the training and testing code is displayed in this Figure:

```
# Train the models
logreg.fit(X_train, y_train)
nb.fit(X_train, y_train)
svm.fit(X_train, y_train)
rf.fit(X_train, y_train)
dt.fit(X_train, y_train)

# Test the models
logreg_pred = logreg.predict(X_test)
nb_pred = nb.predict(X_test)
svm_pred = svm.predict(X_test)
rf_pred = rf.predict(X_test)
dt_pred = dt.predict(X_test)
```

Figure 3.9:Training and testing Model

6. Evaluation Metrics:

The performance of each algorithm was evaluated using the following metrics:

1. Accuracy: Measures the percentage of correctly identified queries (whether malicious or legitimate).
2. Execution Time: To evaluate the computational efficiency of the algorithms.
3. Precision: This metric calculates the percentage of correct positive predictions (correctly identified malicious queries) out of the total predicted positive queries (malicious queries). It indicates the number of predicted malicious queries that were actually malicious.

4. Recall: Recall measures the percentage of actual positive cases (all malicious queries in the dataset) that are correct positive predictions. It demonstrates how well the algorithm found each relevant case.
5. F1 Score: This is the harmonic mean of precision and recall. When the distribution of classes is unbalanced, it provides a favorable balance between recall and precision. A higher F1 Score indicates superior overall performance in reducing false positives and false negatives while accurately identifying fraudulent queries.
6. Other Metrics: To comprehensively evaluate the performance of each model, several metrics were calculated, such as precision, recall, and F1 score. These metrics helped determine how well the model's reconciled precision and recall, as well as the number of queries correctly identified. I will show a small piece of code that demonstrates how to call these metrics or evaluations:

```
def evaluate_and_store(name, y_true, y_pred, start_time):
    accuracy = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    results.append({
        'Algorithm': name,
        'Accuracy': accuracy,
        'F1 Score': f1,
        'Precision': precision,
        'Recall': recall,
        'Time': time.time() - start_time
    })
```

Figure 3.10:Evaluation Metrics

7. Tools and Libraries

Python and associated libraries like Scikit-learn, TensorFlow, Keras, and Genism were used to carry out the practical portion. Furthermore, the calculations were carried out on a system with the hardware capacity to effectively handle deep learning models, and both datasets were saved in a CSV file.

8. Analysis and Comparison

I used the two routing strategies to train and test all models on both datasets, and then I examined the results to see which strategy best balanced execution time and accuracy. The difference in performance between the two datasets and the impact of the selected routing technique on the algorithms' flexibility were factors I considered during the investigation.

4. Chapter Four: Model Design and Results Evaluation

4.1 Model Design

This chapter outlines the useful process for creating and assessing models to identify SQL injection attacks. Using the following image as a guide, it explains every action you made in practice, from loading and preprocessing the dataset to training and assessing the model and system specifications:

Figure 4.1 : Data Analysis and Model Training Steps

4.1.1 Data Collection

To ensure comprehensive and reliable results, two datasets were utilized in this study:

1. I obtained the first dataset from a site called Kaggle, which contains approximately 5000 SQL queries divided between malicious and benign
2. I obtained the second dataset from a site called Protego, which contains approximately 100,000 SQL queries divided between malicious and benign, and I randomly selected 5000 of them.

The availability, labeling quality, and coverage of various SQL injection patterns of these datasets led to their selection. To eliminate duplicates, inconsistencies, and incomplete records, preprocessing was done on both datasets.

Steps for Data Collection and Analysis:

1. Downloaded datasets from verified sources (Kaggle, Protego).
2. Verified data integrity by checking file structure and format.
3. Performed exploratory data analysis (EDA) to understand the distribution of legitimate and malicious queries.

4. Combined and standardized datasets to create a unified structure for further processing.

-Ethical Considerations: The study's datasets came from Protego and Kaggle, two respectable open-source stages. These stages are eminent for making information accessible to the open for academic and investigate reasons. Vitally, no delicate or actually identifiable data (PII) was included in any of the pre-cleaned datasets. All delicate data, counting IP addresses, timestamps, and client IDs, was killed some time recently being posted to these stages. Subsequently, the data utilized in this consider.

4.1.2 Data Preprocessing

The datasets used in this study were preprocessed and structured prior to acquisition from open-source platforms. Therefore, no additional text cleaning, stop word removal, or normalization techniques—such as lemmatization—were applied during this research. The data was ready for use and maintained the syntactic and lexical characteristics of SQL injection patterns essential for detection.

4.1.2.1 Features of Preprocessing

Despite the fact that the original suppliers preprocessed the data, attention was made to maintain important SQLi markers including structural patterns and special characters (e.g., ', ', --, 1=1). In order for machine learning algorithms to differentiate between benign and malicious queries, these components are essential.

-Vectorization

The following vectorization approaches were used to convert the textual data into numerical representations appropriate for machine learning models:

1. Count Vectorizer: Based on the frequency of each token in the query, it represented the inquiries as sparse matrices.

2. TF-IDF (Term Frequency–Inverse Document Frequency): Weighted the significance of tokens by considering how as often as possible they show up over all questions, making a difference to diminish the effect of commonly happening but less instructive words (Ghozali et al., 2022; Li & Zhang, 2019).
3. Pre-trained Word2Vec Embeddings: Captured semantic connections between terms utilizing thick vector representations that reflect relevant closeness(Xia et al., 2024).

-Dimensionality Reduction

Principal Component Analysis (PCA): Connected to decrease the include space dimensionality, progressing preparing effectiveness whereas protecting key data within the dataset.

4.1.3 Algorithm Selection

A wide run of algorithms was chosen based on their known viability in content classification and inconsistency location. This included traditional machine learning models, progressed deep learning architectures, and hybrid models combining the strengths of both.

Reasons for Choosing These Algorithms:

1. Traditional Algorithms (e.g., Naïve Bayes, SVM):
    1. Efficient on smaller datasets and provide interpretable results.
    2. Proven performance in previous text classification studies (Krishnan et al., 2021).
2. Deep Learning Models (e.g., LSTM, CNN):
    1. Capable of capturing complex patterns and dependencies in sequences.
    2. Demonstrated high accuracy in recent research on SQLi detection (Tang et al., 2020; Zhang et al., 2022).
3. Hybrid Models (e.g., FFNN + Naïve Bayes, FFNN + SVM, FFNN + Random Forest, NN + SVM):

1. Hybrid models combine the qualities of traditional machine learning algorithms with deep learning approaches to overcome the impediments of person models.
2. These models point to improve execution by leveraging the interpretability of machine learning calculations nearby the design acknowledgment capabilities of neural systems.
3. For case, FFNN + Naïve Bayes can use the quick preparing speed of Naïve Bayes in conjunction with the non-linear decision-making capacity of FFNNs, whereas FFNN + SVM can give a vigorous arrangement for classifying complex designs.
4. The hybrid approach is particularly useful when dealing with SQLi detection, where both feature extraction and classification require high accuracy and reliability(Bakush, 2024; Krishna et al.; Luo et al., 2019).

-Limitations and Mitigation

It is crucial to take into account each algorithm's unique restrictions as well as the methods used to address them when assessing how well the chosen algorithms work. The difficulties that traditional machine learning models, deep learning models, and hybrid models encounter are outlined here, along with the methods that have been employed to overcome them(Alghawazi et al., 2022).

Table 4.1 : Summary of Limitations and Mitigation Strategies for ML, DL, and Hybrid Models

Type	**Limitations**	**Mitigation Strategies**
Traditional ML	Struggles with capturing semantic context	Used n-grams and TF-IDF to enhance contextual learning
Deep Learning	High computational demands and overfitting risks on small data	Applied dropout layers, early stopping, and hyperparameter tuning

Hybrid Models	May suffer from increased complexity and training time	Balanced model architecture and regularization techniques

4.1.4 Data Splitting

The remaining 20% of the dataset was used for testing, while the remaining 80% was used for preparation. The train test split function from scikit-learn was used in conjunction with a settled random seed to verify that the results could be reproduced (Brownlee, 2022; Géron, 2019).

-An 80-20 Split: Why Use It?

The 80/20 ratio creates a pleasing equilibrium. It maintains enough samples to fully evaluate how well the models perform on unknown data while providing enough data to train the models efficiently. Stratified sampling was employed to maintain class balance in both the training and testing sets in order to guarantee equity. Below this figure Show code to split data (80-20):

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 4.2:Split Dataset

Adjusting the Settings:

A number of parameters were examined in order to determine the ideal model settings, including:
Batch size: Tested values of 16, 32, and 64.
Number of epochs: Ranged from 10 to 30.
Optimizers: Compared Adam, SGD, and RMSProp.
Test size: Tried different splits, such as 0.2, 0.3, and 0.4.
This process of testing and adjusting these parameters (called sensitivity analysis) helped determine the optimal combination of settings for each model.

4.1.5 Model Training and Evaluation

-All models were assessed utilizing five key metrics:
Accuracy, Precision, Recall, F1-Score, and Execution Time. These measurements were chosen to supply a comprehensive assessment of both discovery quality and computational proficiency.

-Evaluation Metrics Explained

1. Accuracy: Measures overall correctness, but may be misleading in imbalanced datasets.
2. Precision: Helps reduce false positives, minimizing unnecessary alerts.
3. Recall: Essential in security systems to avoid missing real threats (false negatives).
4. F1-Score: Balances precision and recall, especially useful when both false positives and false negatives are critical.
5. Execution Time: Indicates the feasibility of using the model in real-time environments (Marwah et al., 2024; Yacouby & Axman, 2020).

-Model-Specific Configurations

Each model was prepared utilizing carefully chosen hyperparameters to maximize performance. A full summary of the settings and evaluation results can be found in the images below. Highlights incorporate:

1- Logistic Regression

```
# Logistic Regression
start_time = time.time()
clf = LogisticRegression(random_state=42)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
results.append(('LR', accuracy))
print("LR - Accuracy: {:.2f}, Time: {:.2f} seconds".format(accuracy * 100, time.time() - start_time))
```

Figure 4.3:LR CODE

- Solver: lbfgs, Max Iterations: 100.
- Focused on improving recall to avoid undetected threats.

2-Decision Tree

```
# Decision Tree
start_time = time.time()
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_train)
y_pred = dt_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
results.append(('Decision Tree', accuracy))
print("Decision Tree - Accuracy: {:.2f}, Time: {:.2f} seconds".format(accuracy * 100, time.time() - start_time))
```

Figure 4.4 : Decision Tree

- Max Depth: 10, Criterion: Gini.
- Used normalization for more balanced decision boundaries.

3-Random Forest

```
# Random Forest
start_time = time.time()
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
results.append(('Random Forest', accuracy))
print("Random Forest - Accuracy: {:.2f}, Time: {:.2f} seconds".format(accuracy * 100, time.time() - start_time))
```

Figure 4.5 : Random Forest Code

- Estimators: 100, Max Depth: 10.
- Prioritized ensemble robustness and accuracy.

4-Feedforward Neural Network (FFNN)

```
# Feedforward Neural Network
start_time = time.time()
ffnn_model = Sequential()
ffnn_model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
ffnn_model.add(Dense(32, activation='relu'))
ffnn_model.add(Dense(1, activation='sigmoid'))
ffnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
ffnn_model.fit(X_train, y_train, epochs=10, batch_size=16, verbose=1)
_, accuracy = ffnn_model.evaluate(X_test, y_test)
results.append(('FFNN', accuracy, None, None, time.time() - start_time))
print("FFNN - Accuracy: {:.2f}, Time: {:.2f} seconds".format(accuracy * 100, time.time() - start_time))
```

Figure 4.6 : FFNN Code

- Batch Size: 16, Epochs: 10, Optimizer: Adam.
- Included dropout layers, batch normalization, and embedding for better generalization.

5-XGBoost

```
# XGBoost
start_time = time.time()
xgb_classifier = xgb.XGBClassifier()
xgb_classifier.fit(X_train, y_train)
y_pred = xgb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
results.append(('XGBoost', accuracy))
print("XGBoost - Accuracy: {:.2f}, Time: {:.2f} seconds".format(accuracy * 100, time.time() - start_time))
```

Figure 4.7 : XGBoost Code

- Learning Rate: 0.1, Max Depth: 6.
- Tuned for fast convergence with high precision.

6-Naïve Bayes / SVM / AdaBoost / Passive Aggressive

```
# Naive Bayes
start_time = time.time()
nb_classifier = GaussianNB()
nb_classifier.fit(X_train, y_train)
y_pred = nb_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
results.append(('Naive Bayes', accuracy))
print("Naive Bayes - Accuracy: {:.2f}, Time: {:.2f} seconds".format(accuracy * 100, time.time() - start_time))
```

Figure 4.8 : Naive Bayes

- Customized settings to enhance text classification performance

7-Hybrid Model: FFNN + SVM

```
# Hybrid FFNN + SVM
def hybrid_model_ffnn_svm(X_train, X_test, y_train, y_test):
    start_time = time.time()
    model = Sequential([
        Dense(64, activation='relu', input_dim=X_train.shape[1]),
        Dense(32, activation='relu'),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    model.fit(X_train, y_train, epochs=10, batch_size=16, verbose=1)

    feature_train = model.predict(X_train)
    feature_test = model.predict(X_test)

    svm = SVC()
    svm.fit(feature_train, y_train)
    y_pred = svm.predict(feature_test)
    evaluate_model('Hybrid FFNN + SVM', y_test, y_pred, start_time)
```

Figure 4.9 : Hybrid FFNN + SVM Code

- Combined FFNN's feature extraction with SVM's robust classification
- Trained FFNN to produce deep feature vectors which were fed into a linear SVM
- Aimed to maximize F1-Score and recall for high-risk SQL injection patterns (Liu et al., 2021).

4.1.6 System Specifications and Configuration Details for Testing Code

A personal laptop was used for both the code execution and testing. To ensure reproducibility and draw attention to the computational resources needed to carry out the machine learning computations, it is essential to indicate the framework hardware and program configuration. The detailed findings of the testing environment are listed below:

Table 4.2 : component of system

Component	**Specification**
Operating System	Windows 10 Pro
Processor (CPU)	Intel(R) Core (TM) i7-7500U CPU @ 2.70GHz 2.90 GHz
RAM (Memory)	8.00 GB

Storage	512 GB SSD
Programming Tools	Python 3.12.7, packaged by Anaconda
IDE	Jupyter Notebook, integrated with Anaconda
GPU	Intel(R) HD Graphics 620

Additionally, the following Python libraries and frameworks were utilized during the implementation and testing phases:

- Scikit-learn for machine learning algorithms.
- Pandas and NumPy for data preprocessing and manipulation.
- Matplotlib and Seaborn for visualizations.

Hardware Details:

Training large-scale deep learning models, especially those using high-dimensional data like LSTM, CNN, and Bi-LSTM, was difficult due to the lack of a dedicated GPU and the 8GB RAM constraint. The following tactics were used to lessen these problems and preserve model performance:

-Using smaller batch sizes (e.g., 16 or 32).
-Reducing the number of epochs for deep neural networks.
-Applying dimensionality reduction techniques.
-Leveraging optimized data pipelines for loading and preprocessing.

Although these limitations can potentially affect the models' flexibility, the resources used were sufficient to run, evaluate, and contrast the performance of the selected algorithms for research. Future research might look into using cloud-based computing platforms or powerful GPUs to increase adaptability and productivity(Chen et al., 2020).

-Tools for Software Implementation:

Jupyter Notebook was chosen for its intuitively environment, consistent integration with logical Python libraries, and adaptability in iterative testing. The utilize

of Boa constrictor rearranged bundle administration and form control, helping in keeping up consistency all through the project.

-Libraries and Frameworks Used:

The usage depended on a assorted set of Python libraries and systems for machine learning, deep learning, and visualization errands. The key libraries utilized, together with their adaptations (where appropriate), are as takes after:

1. Scikit-learn (v1.5.1): For traditional machine learning models and evaluation metrics
2. Pandas (v2.2.2): For data manipulation and analysis.
3. NumPy (v1.26.4): For numerical operations and array processing.
4. Matplotlib (v3.9.2) and Seaborn (v0.13.2): For data visualization and result comparison.
5. TensorFlow/Keras: For implementing and training deep learning models (e.g., FFNN, LSTM, Bi-LSTM, CNN, RNN).
6. XGBoost: For gradient boosting model implementation.
7. GridSearchCV (from Scikit-learn): For hyperparameter tuning and optimization.

Additional components from the code include:

-Tokenization and Padding: Tokenizer, pad_sequences from TensorFlow for preparing text data for deep learning:

1. Callbacks: Early Stopping for preventing overfitting.
2. Optimizers: Adam, RMSprop for model training.
3. Metrics: accuracy score, precision score, recall score, and f1_score for performance evaluation.

4.1.7 Hardware Limitations and Computational Constraints

We conducted the tests in this investigation on a standard individual portable workstation, which had an 8GB RAM setup and no dedicated GPU. These equipment

restrictions forced imperatives on show complexity, preparing time, and hyperparameter tuning. For the occasion, profound learning models such as FFNN, RNN, and LSTM were prepared utilizing moderately little clump sizes (e.g., 16 or 32) and a restricted number of ages (e.g., 10) to maintain a strategic distance from memory flood and over-the-top computational time. These restrictions too limited the utilization of more progressed methods such as network rummaging around for hyperparameter optimization or the integration of expansive relevant embeddings like BERT. As a result, demonstrated designs and prepared arrangements were chosen with an accentuation on productivity and possibility inside obliged assets. Future work may take advantage of high-performance computing situations to investigate more profound designs and broader parameter tuning(Liu et al., 2024; Sze et al., 2017).

4.2 Results and Evaluation

4.2.1 Dataset Descriptions

Dataset 1: Kaggle SQL Injection Dataset
Source: Taken from Kaggle .
Composition: Contains two columns: one labeled "SQL queries" (1 for injection, 0 for clean), and the other for their related labels.
Volume: 4,201 rows are present.
Queries:

1. Statement Queries (Label=0): Legitimate SQL queries that are typically used to access or modify data. Their percentage in the dataset is here 73.14%.
2. Non-statement queries (Label=1): Queries that are intended to exploit vulnerabilities, and are used in SQL injection attacks to allow attackers to compromise a database or access unauthorized data. Their percentage in the dataset is here 26.86%(Shah, 2024).

Dataset 2: Protego SQL Injection Dataset
Source: Taken from Protego .
Composition: consists of two columns with labels (1 for injection and 0 for clean) combined with SQL queries.

Original Volume: 100,000 rows.
Sampled Volume: Because the original dataset was so big, a random sample of 5,000 rows was chosen.
Queries:

1. Statement Queries (Label=0): Legitimate SQL queries that are typically used to access or modify data. Their percentage in the dataset is here 50.16%.
2. 2-Non-statement queries (Label=1): Queries that are intended to exploit vulnerabilities, and are used in SQL injection attacks to allow attackers to compromise a database or access unauthorized data. Their percentage in the dataset is here 49.84%(*SQL Injection dectection payloads*, 2022).

I collected my dataset and code on Kaggle, here is the link.

4.2.2 Feature Weight Extraction using Machine Learning Algorithms

Each algorithm has a method or approach to extract features from datasets; I will show the top 5 features of some algorithms and their value (importance):

Dataset1:
1-XGBoost

Table 4.3 : Top 5 Features XGBoost

Feature	**Weight**
select	0.332972
admin	0.040791
quot	0.029896
apos	0.027567
__time__	0.026208

2-Random Forest

Table 4.4 : Top 5 Features Random Forest

Feature	Weight
select	0.142053
id	0.081376
users	0.081340
version	0.044449
admin	0.037283

3-Decision Tree

Table 4.5 : Top 5 Features Decision Tree

Feature	Weight
select	0.461154
admin	042971
exec	0.025026
__time__	0.022823
hi	0.017952

4-Logistic Regression

Table 4.6 : Top 5 Features Logistic Regression

Feature	Weight
select	3.025876
admin	2.781934
hi	2.106002
quot	1.991559
exec	1.893620

5-SVM

Table 4.7 : Top 5 Features SVM

Feature	Weight
life	1.230341
billion	1.230178

man	1.230176
dr	1.230143
com	1.230132

Dataset2:

1-XGBoost

Table 4.8 : Top 5 Features XGBoost

Feature	**Weight**
rows	0.0614
vyh	0.0544
benchmark	0.0472
elt	0.0451
select	0.0434

2-Random Forest

Table 4.9 : Top 5 Features Random Forest

Feature	**Weight**
union	0.1046
select	0.0690
sleep	0.0477
vyhv	0.0337
null	0.0291

3-Decision Tree

Table 4.10 : Top 5 Features Decision Tree

Feature	**Weight**
select	0.2348
sleep	0.0753
elt	0.0490
null	0.0465
case	0.0387

4-Logistic Regression

Table 4.11 : Top 5 Features Logistic Regression

Feature	**Weight**
sleep	4.0741
Union	4.0365
elt	3.0726
pg_sleep	3.0057
make_set	2.8807

5-SVM

Table 4.12 : Top 5 Features SVM

Feature	**Weight**
union	0.6226
vyhv	0.5410
end	0.4797
sleep	0.4696
like	0.4598

4.2.3 Experimental Results

Key evaluation measures like accuracy, precision, recall, F1-score, and execution time are highlighted in this part, which is divided into several subsections. Each algorithm's performance across the various vectorization techniques—Word2Vec, Count Vectorizer, and TF-IDF—is shown in a distinct figure for clarity's sake. It is simpler to compare and examine the outcomes according to algorithm and vectorization technique because each figure incorporates all five-assessment metrics:

1. Using Word Vectorizer.

2. Using Count Vectorizer.
3. Ussing TF-IDF.

These vectorizers are two distinct ways to convert text into vectors, with default variable and factor values initially applied. Later, the values of certain parameters (“batch_size, epochs, optimizer, random_state, test_size”) were varied individually to assess their impact on accuracy, keeping other parameters at their default values (e.g., batch_size=32, epochs=10, optimizer=” Adam”, random_state=42, test_size=0.2).

4.2.3.1 Accuracy Results

In this section we show the accuracy rate of each algorithm, and the results were displayed in a common image so that the black column used Word Vectorizer and the green Count Vectorizer, in addition to that I tried to change the values of many parameters to get the best accuracy rate, these parameters are (batch_size, test_size, random state, optimizer, epochs), and these experimental results are available in this link. Then we used the best values until the accuracy rate became the best possible, which is as follows:

1-Dataset1:

After testing, it was determined that the best accuracy rate was achieved using the following parameters in Count Vectorizer: batch_size = 16, epochs = 10, optimizer = 'Adam', random_state = 42, and test_size = 0.2. It was also determined that the best accuracy rate was achieved using the following parameters in Word Vectorizer: batch_size = 32, epochs = 15, optimizer = 'Adam', random_state = 24, and test_size = 0.2. Furthermore, using TF-IDF Vectorizer, the best accuracy rate was obtained with batch_size = 32, epochs = 10, optimizer = 'Adam', random_state = 42, and test_size = 0.2. These configurations represent the highest accuracy rates achieved across the different vectorization techniques. The accuracy rate results for each type of algorithm (traditional and deep learning) are shown in the corresponding figures.
-The first group of Figures shows Performance Metrics of the Traditional Algorithms:

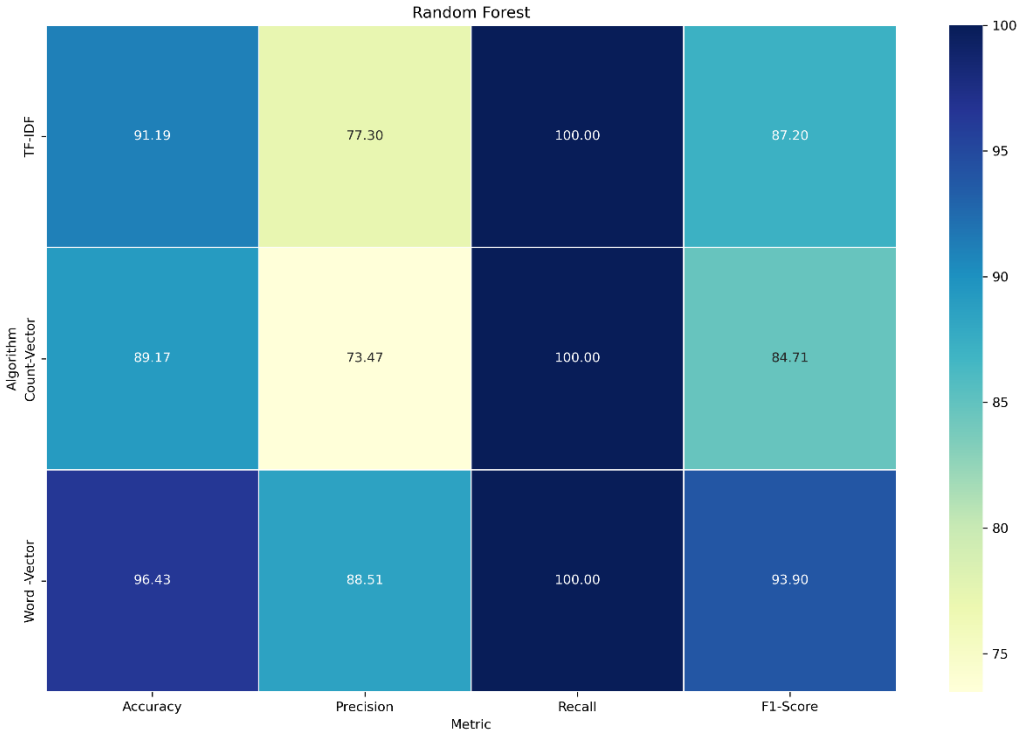


Figure 4.10: Performance Metrics per Random Forest per Dataset1

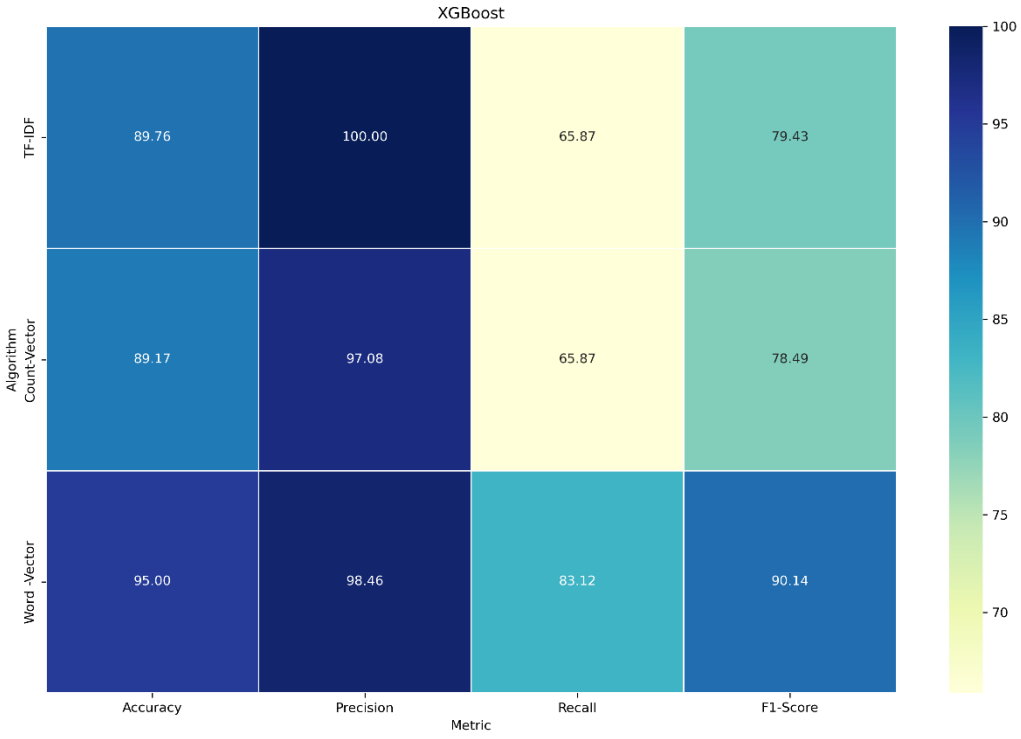


Figure 4.11 : Performance Metrics per XGBoost per Dataset1

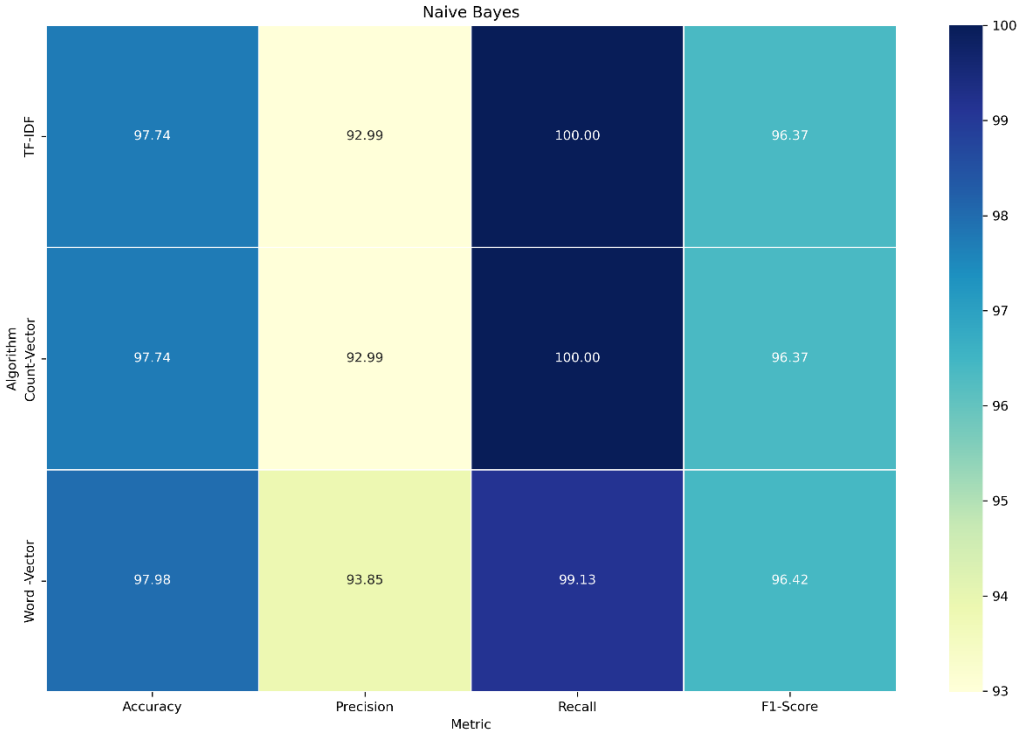


Figure 4.12 : Performance Metrics per Naive Bayes per Dataset1

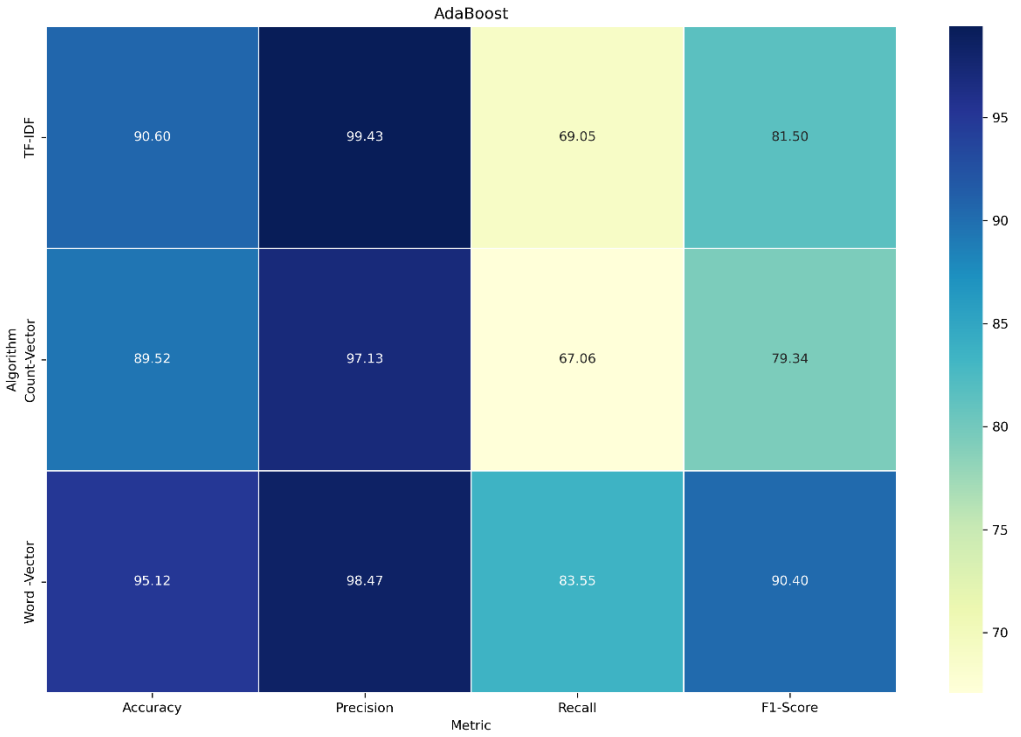


Figure 4.13 : Performance Metrics per AdaBoost per Dataset1

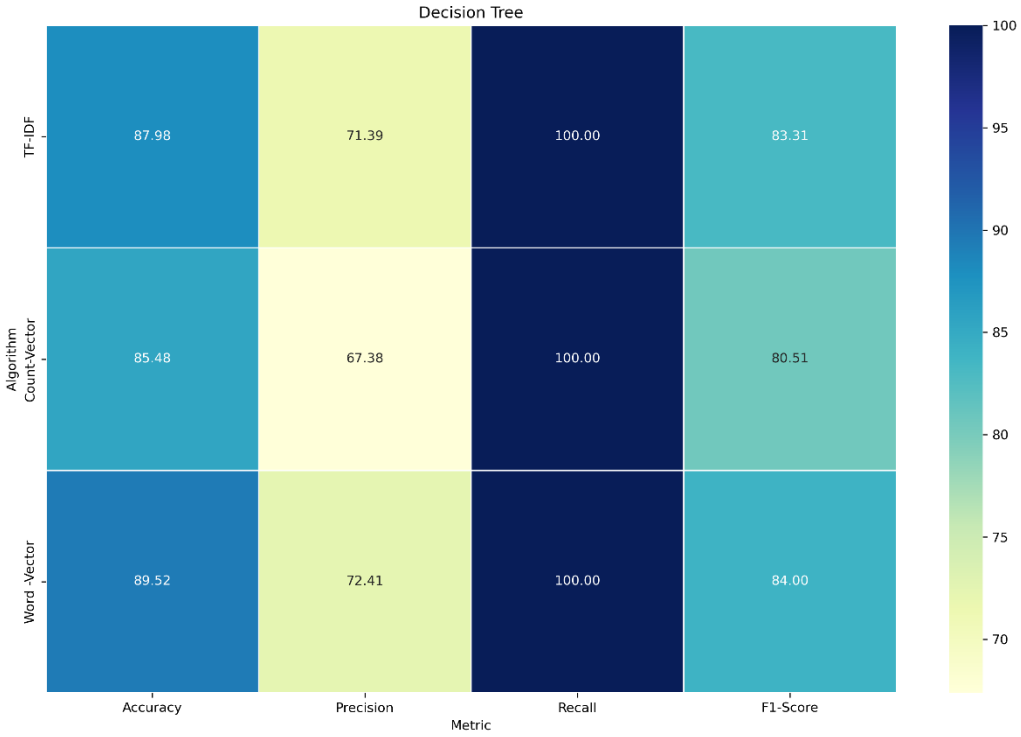


Figure 4.14 : Performance Metrics per Decision Tree per Dataset1

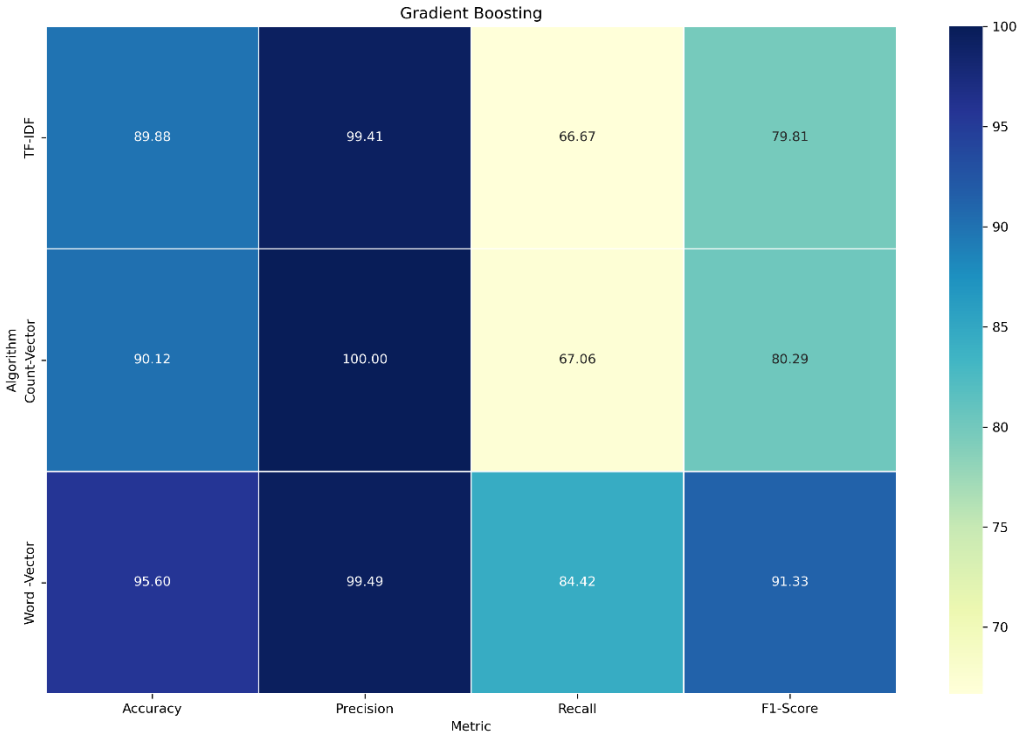


Figure 4.15 : Performance Metrics per Gradient Boosting per Dataset1

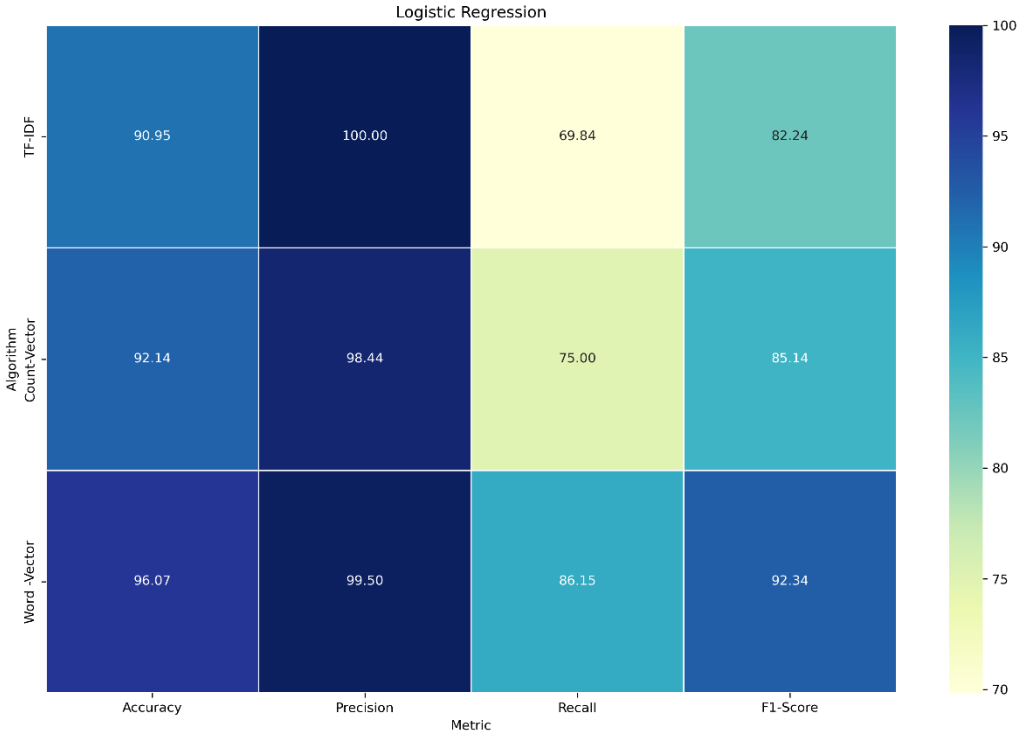


Figure 4.16 : Performance Metrics per Logistic Regression per Dataset1

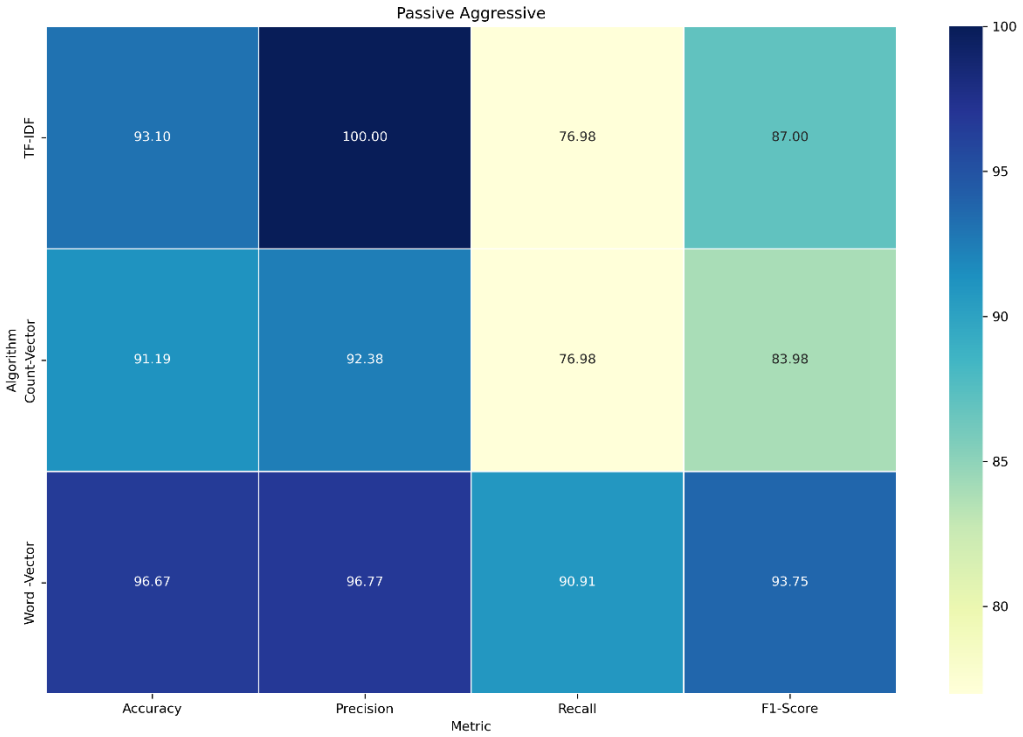


Figure 4.17 : Performance Metrics per Passive Aggressive per Dataset1

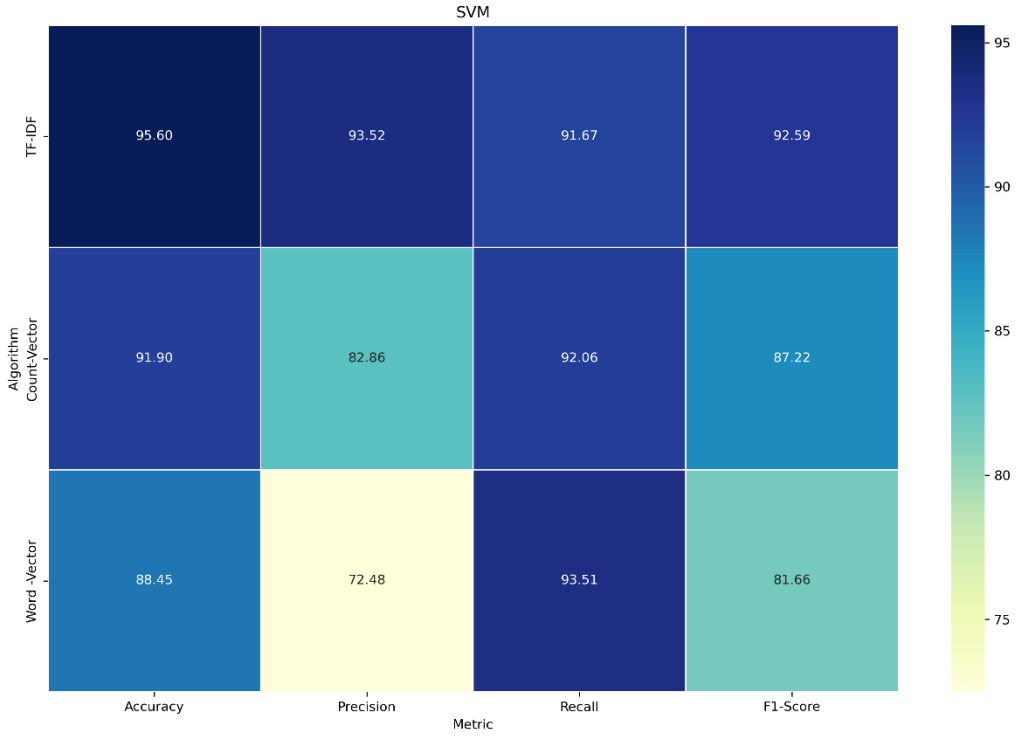


Figure 4.18 :Performance Metrics per SVM per Dataset1

-The second group of Figures shows Performance Metrics of the Deep Learning Algorithms:

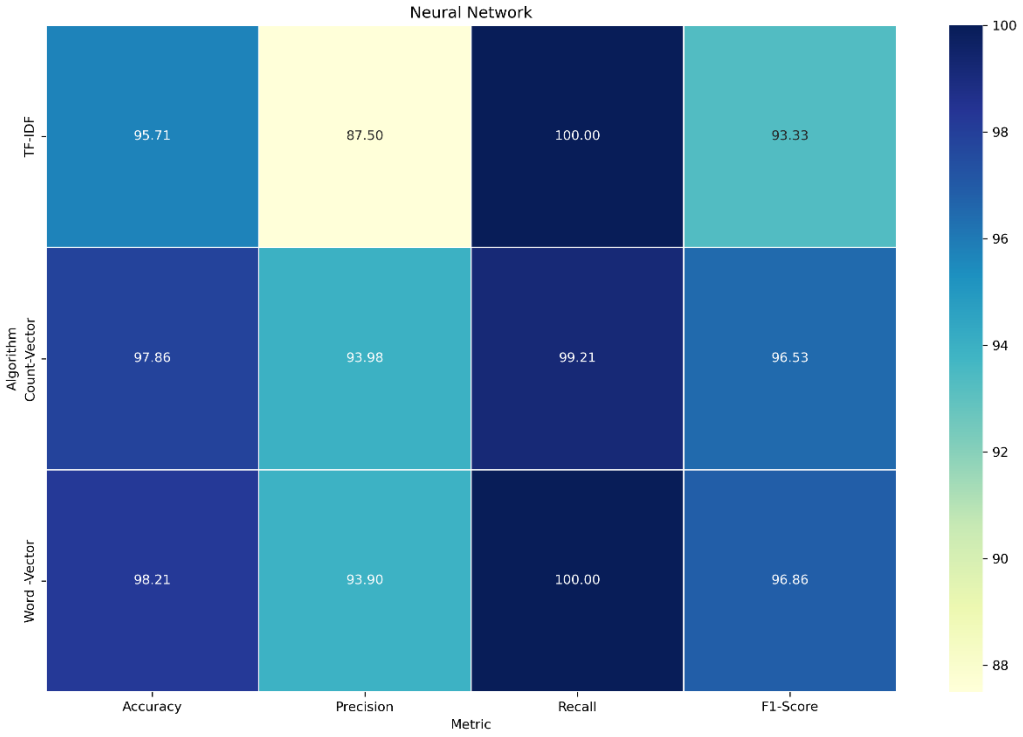


Figure 4.19 : Performance Metrics per Neural Network per Dataset1

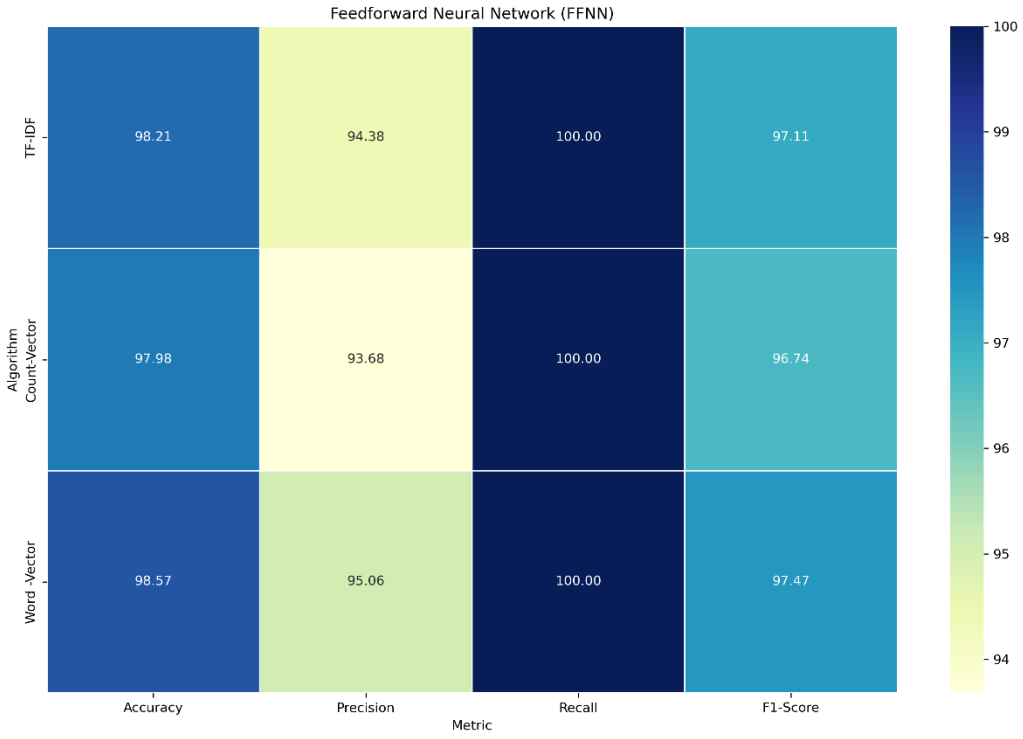


Figure 4.20 : Performance Metrics per Feedforward Neural Network per Dataset1

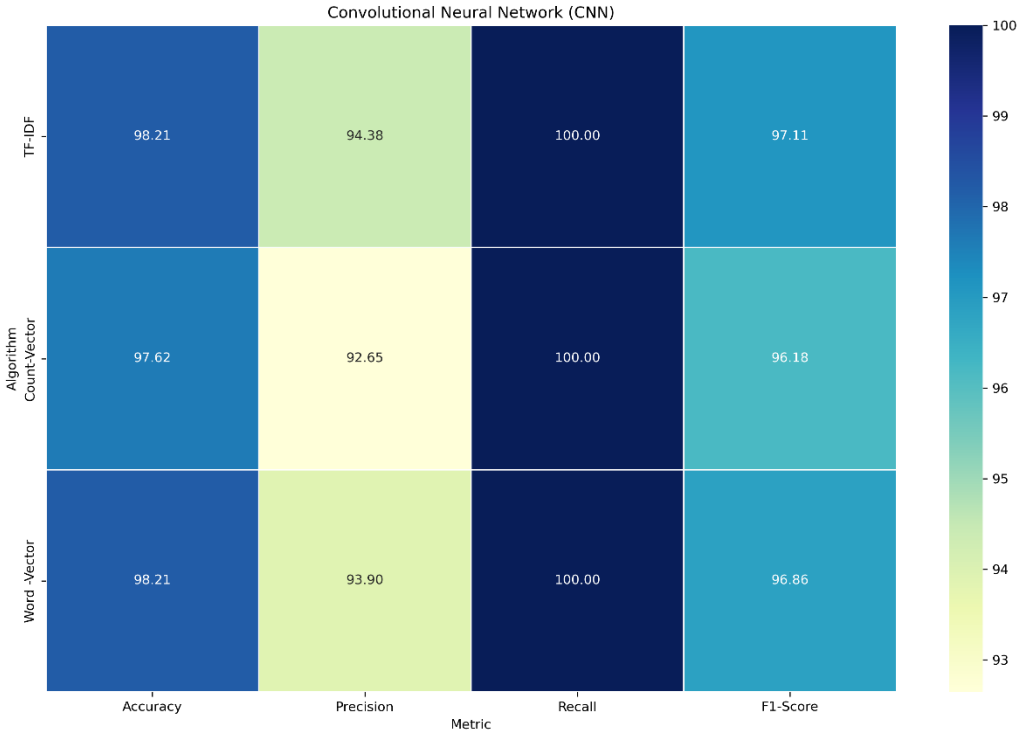


Figure 4.21 : Performance Metrics per CNN per Dataset1

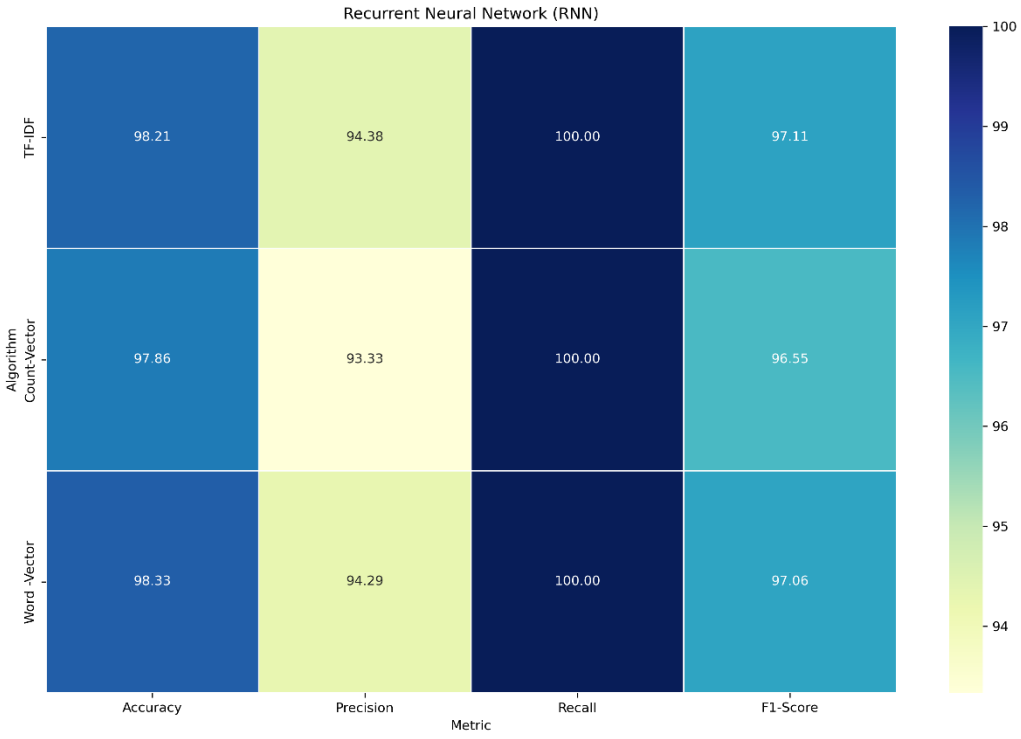


Figure 4.22 : Performance Metrics per RNN per Dataset1

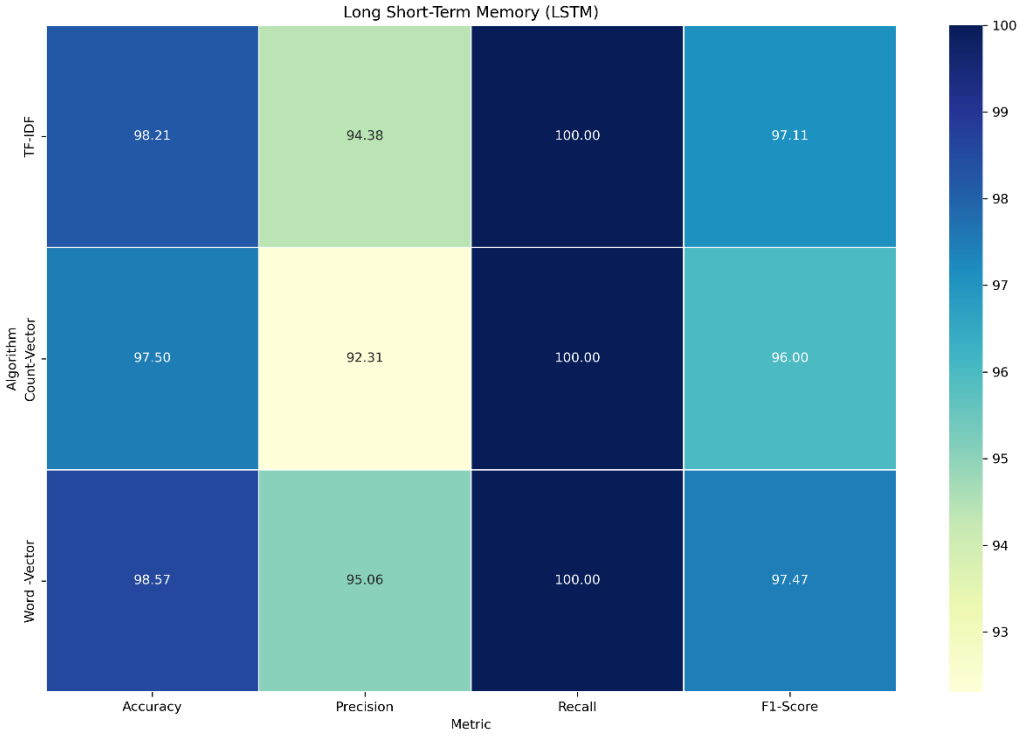


Figure 4.23 : Performance Metrics per LSTM per Dataset1

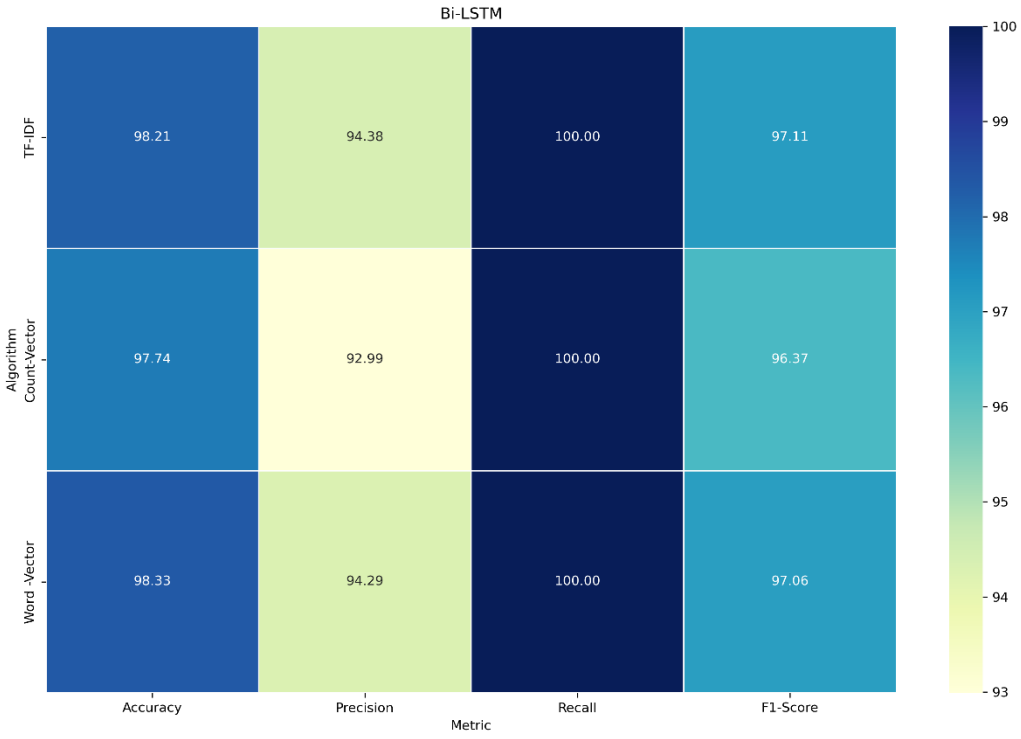


Figure 4.24 : Performance Metrics per Bi-LSTM per Dataset1

-The third group Figures shows the Performance Metrics of the hybrid Algorithms:

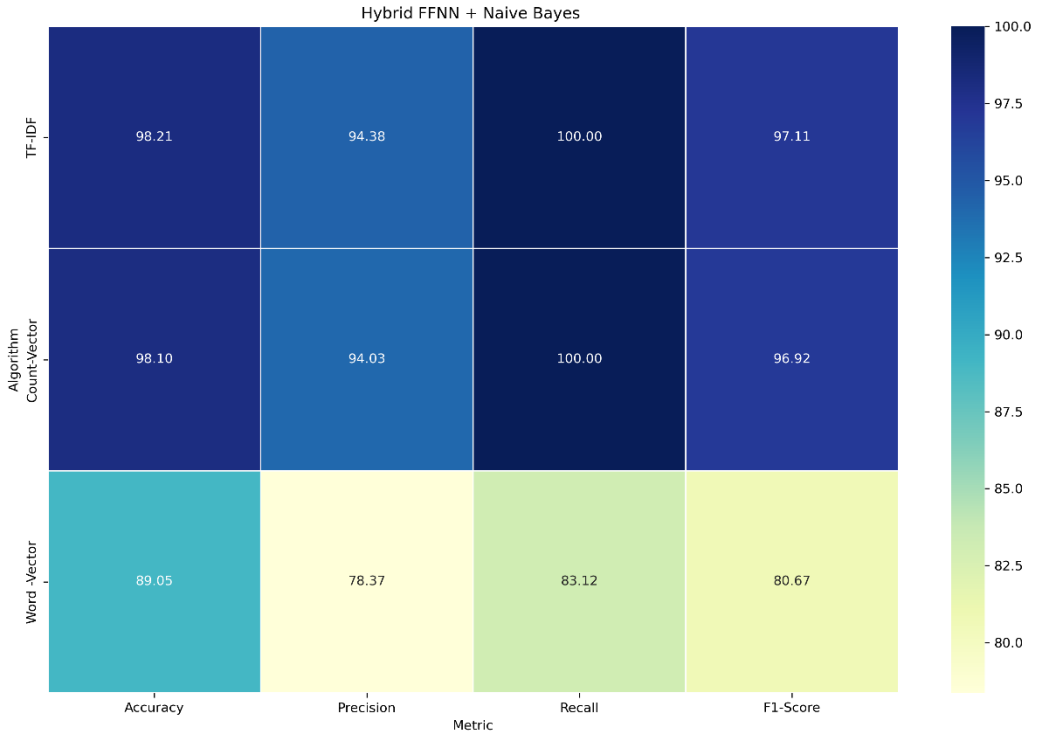


Figure 4.25 : Performance Metrics per Hybrid FFNN + Naive Bayes per Dataset1

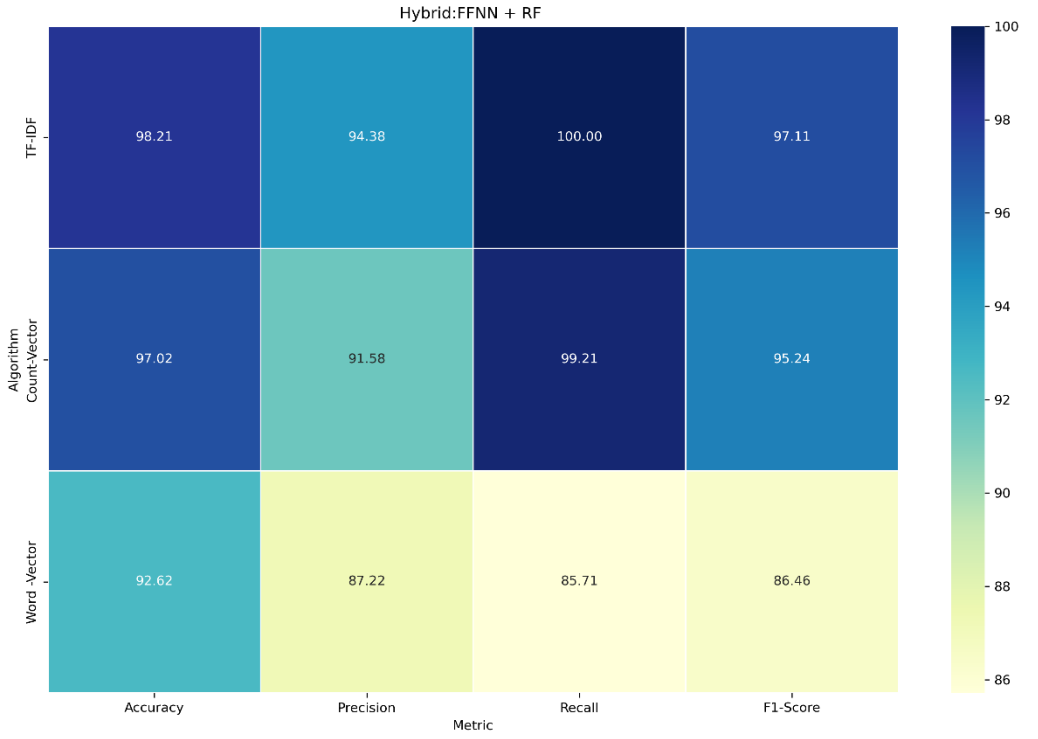


Figure 4.26 : Performance Metrics per Hybrid FFNN + Random Forest per Dataset1

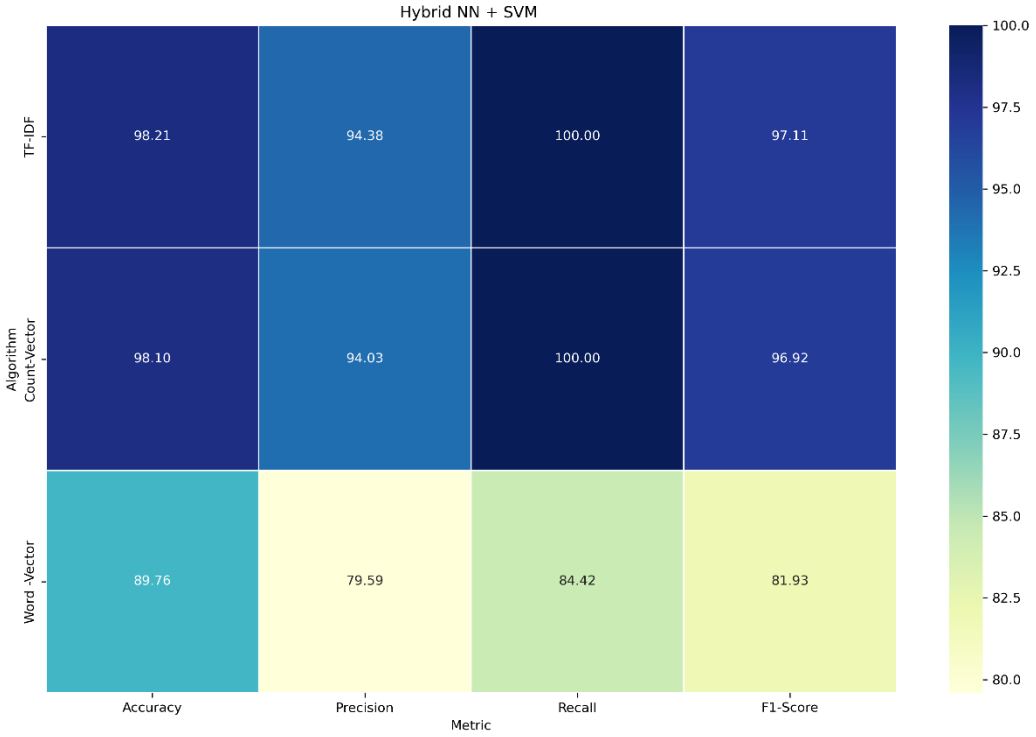


Figure 4.27 : Performance Metrics per Hybrid Neural Network + SVM per Dataset1

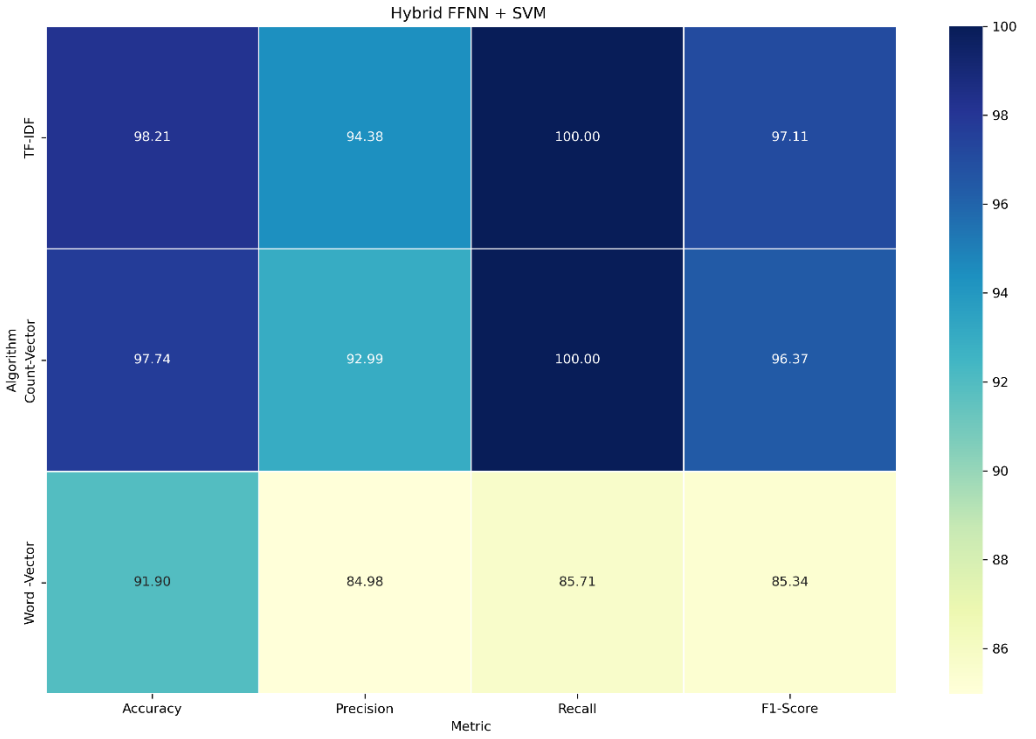


Figure 4.28 : Performance Metrics per Hybrid FFNN + SVM per Dataset1

2-Dataset 2:

After testing, it was determined that the best accuracy rate was achieved using the following parameters in Count Vectorizer: batch_size = 16, epochs = 15, optimizer = 'Adam', random_state = 24, and test_size = 0.4. It was also determined that the best accuracy rate was achieved using the following parameters in Word Vectorizer: batch_size = 32, epochs = 10, optimizer = 'Adam', random_state = 42, and test_size = 0.2, Additionally, using TF-IDF, the optimal performance was obtained with the parameters: batch_size = 32, epochs = 10, optimizer = 'Adam', random_state = 42, and test_size = 0.2. This represents the highest accuracy rate obtained for each technique respectively. The accuracy rate results for each type of algorithm (traditional and deep learning and Hybrid algorithm) are shown in the corresponding figure for each vectorization technique.

-The first group of Figures shows the Performance Metrics of the Traditional Algorithms:

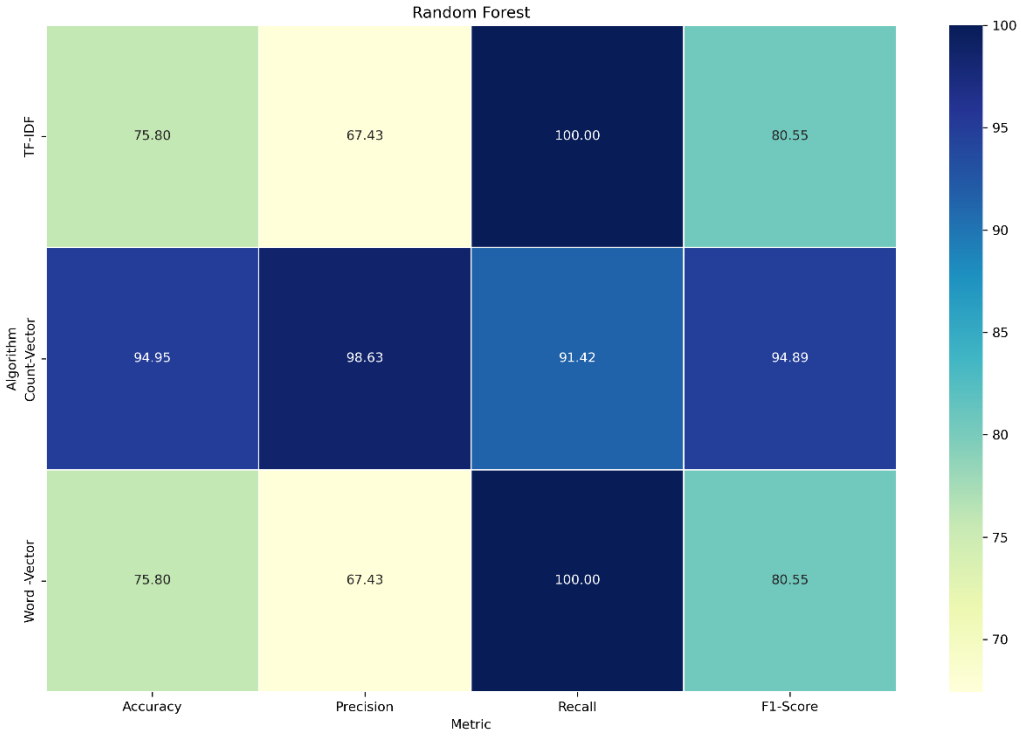


Figure 4.29 : Performance Metrics per Random Forest per Dataset2

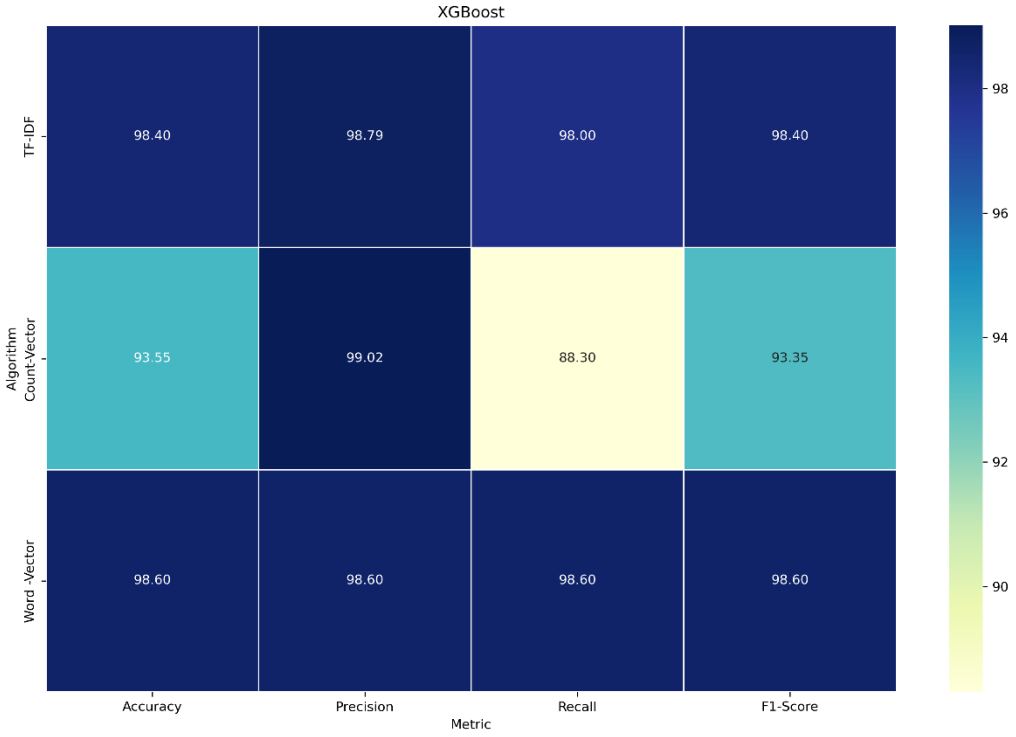


Figure 4.30 : Performance Metrics per XGBoost per Dataset2

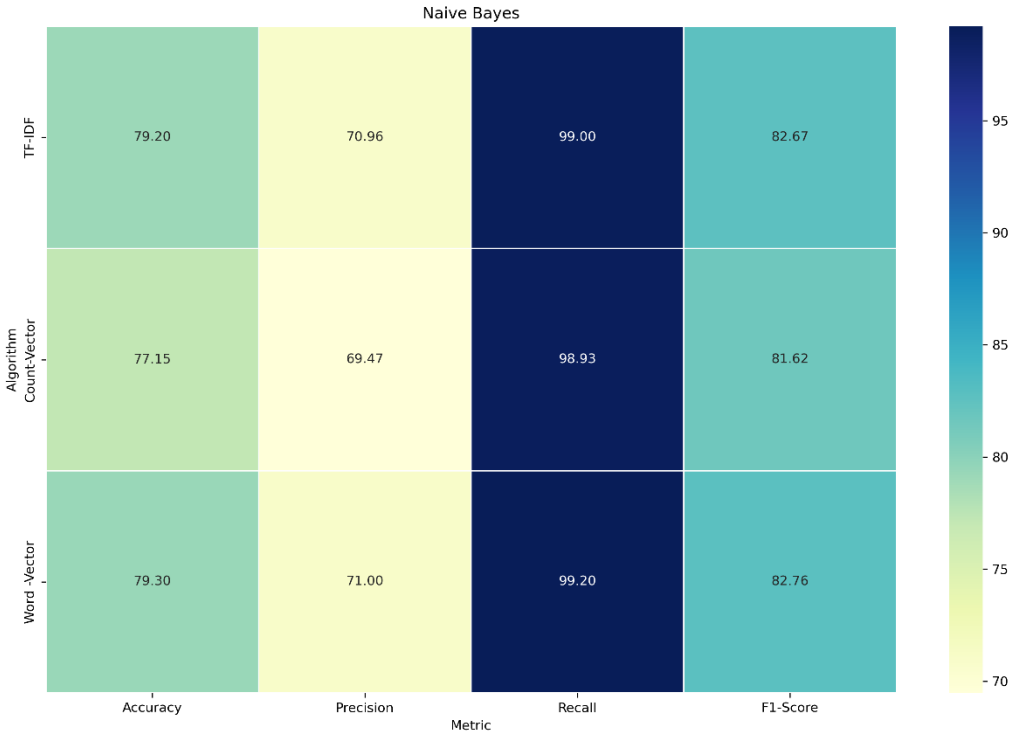


Figure 4.31 : Performance Metrics per Naive Bayes per Dataset2

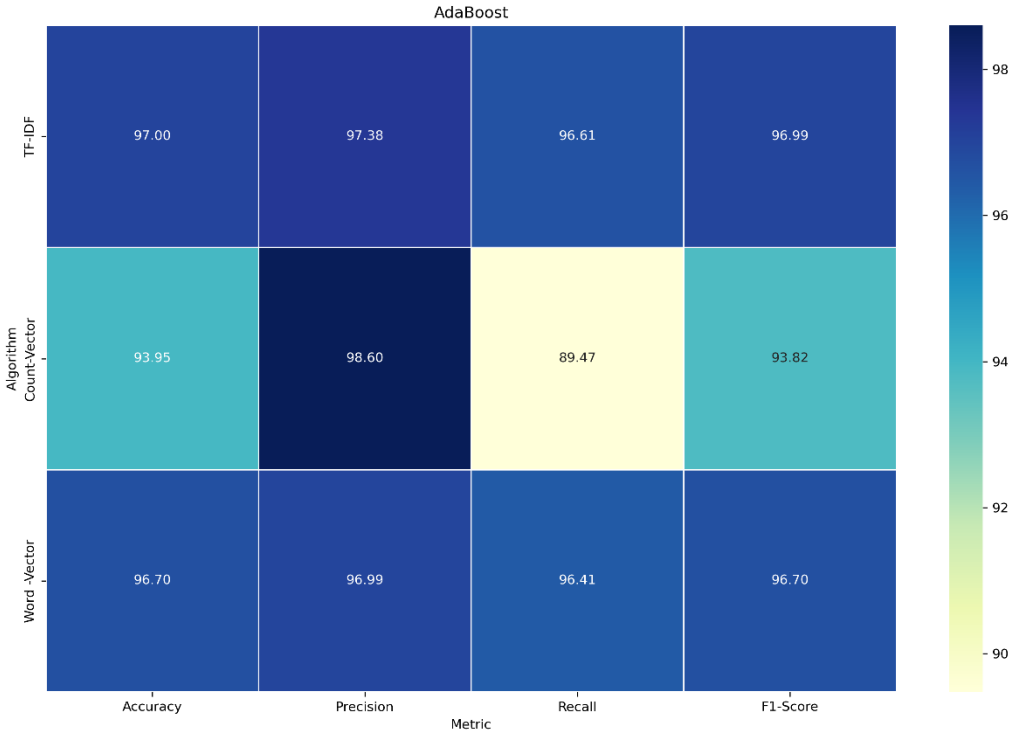


Figure 4.32 : Performance Metrics per AdaBoost per Dataset2

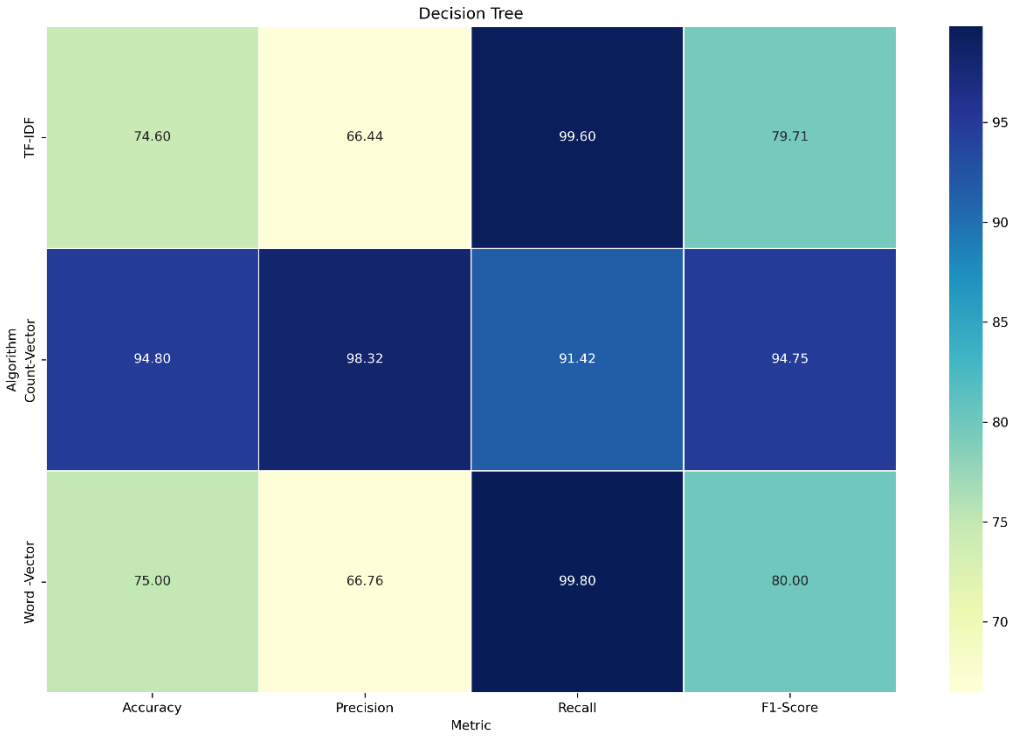


Figure 4.33 : Performance Metrics per Decision Tree per Dataset2

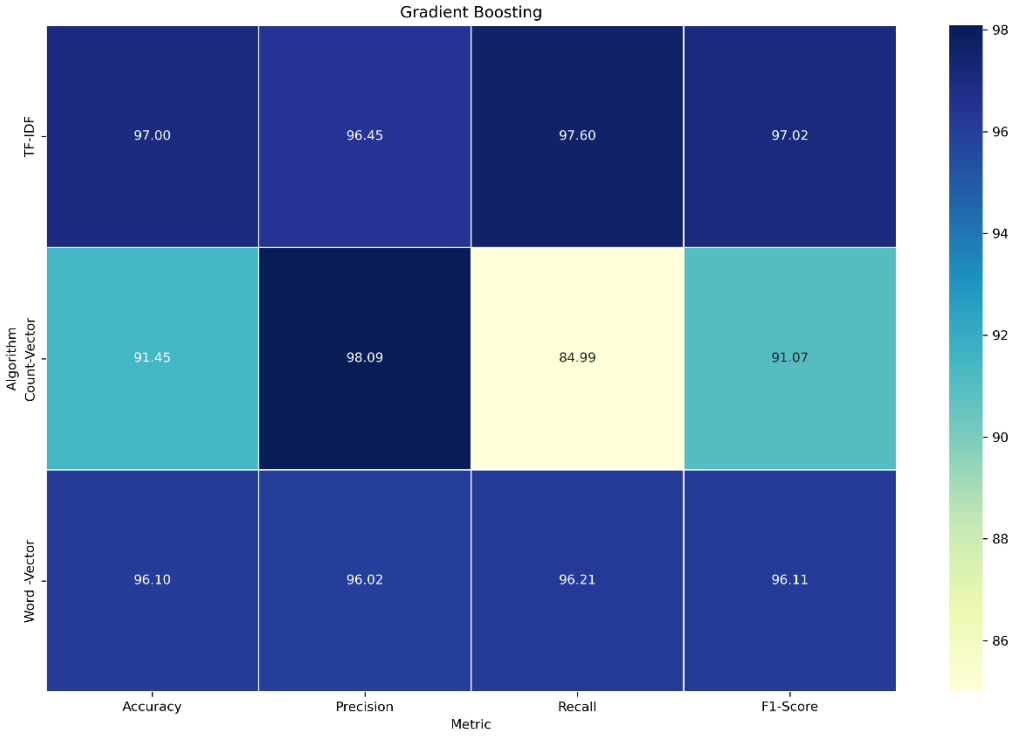


Figure 4.34 : Performance Metrics per Gradient Boosting per Dataset2

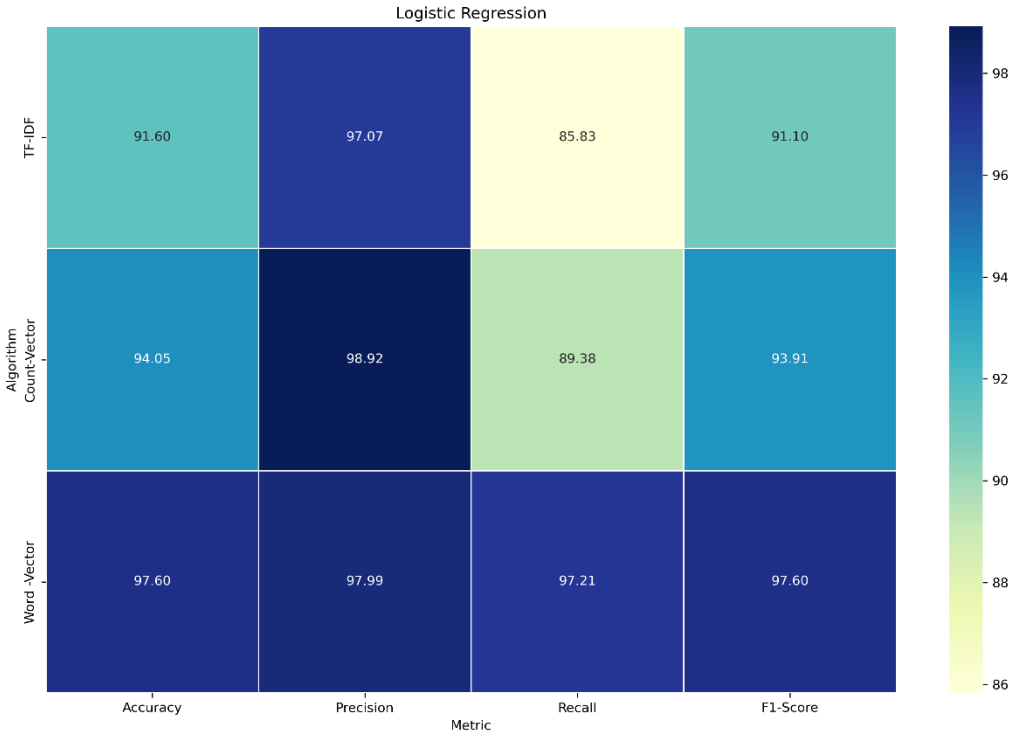


Figure 4.35 : Performance Metrics per Logistic Regression per Dataset2

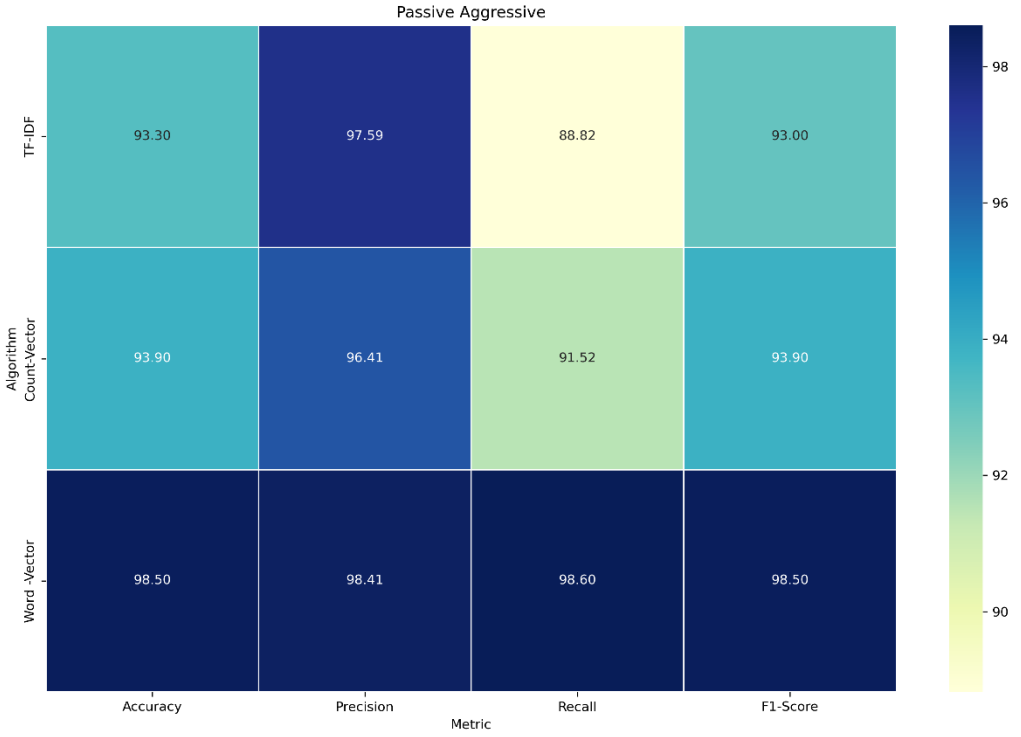


Figure 4.36 : Performance Metrics per Passive Aggressive per Dataset2

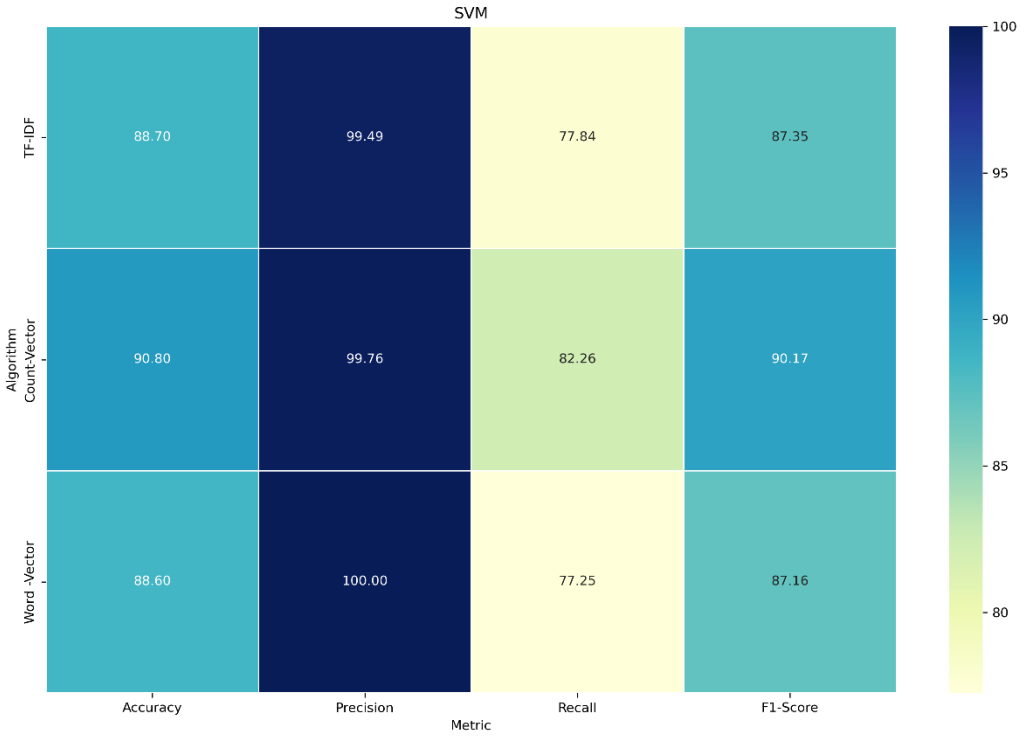


Figure 4.37 : Performance Metrics per SVM per Dataset2

-The second group of Figures shows the Performance Metrics of the Deep Learning Algorithms:

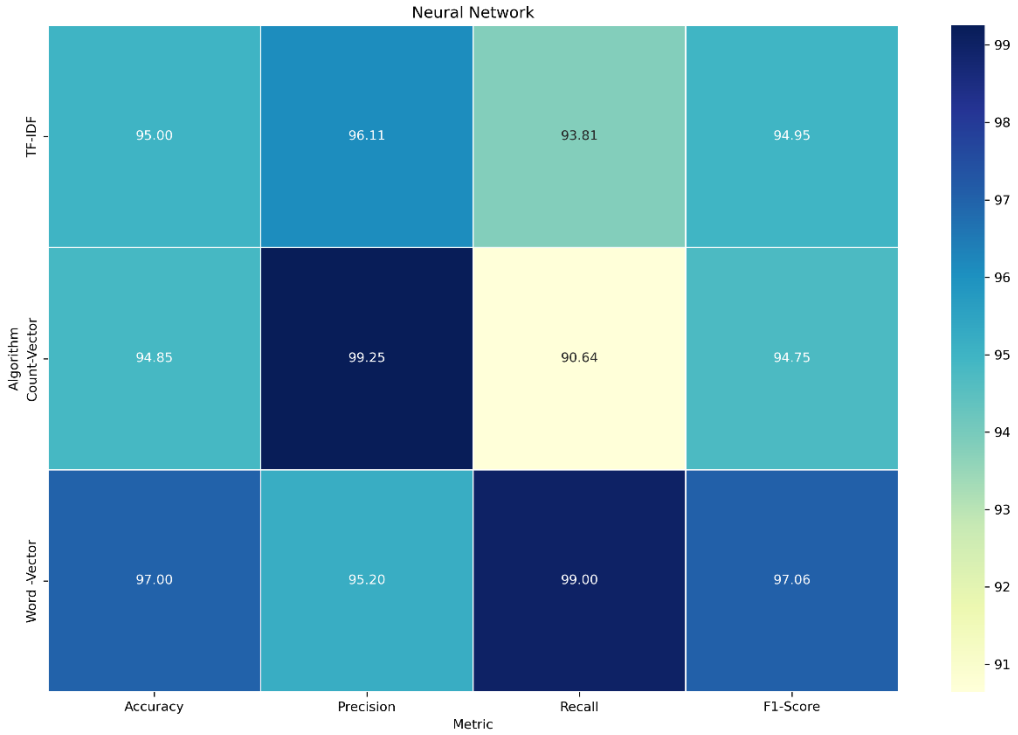


Figure 4.38 : Performance Metrics per Neural Network per Dataset2

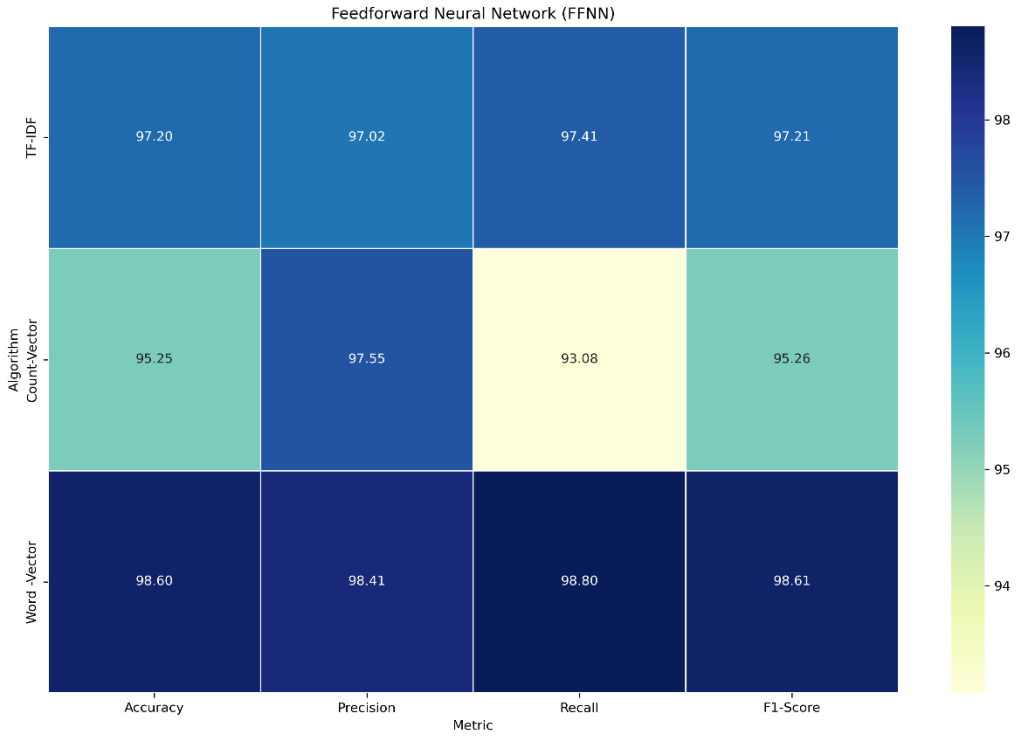


Figure 4.39 : Performance Metrics per FFNN per Dataset2

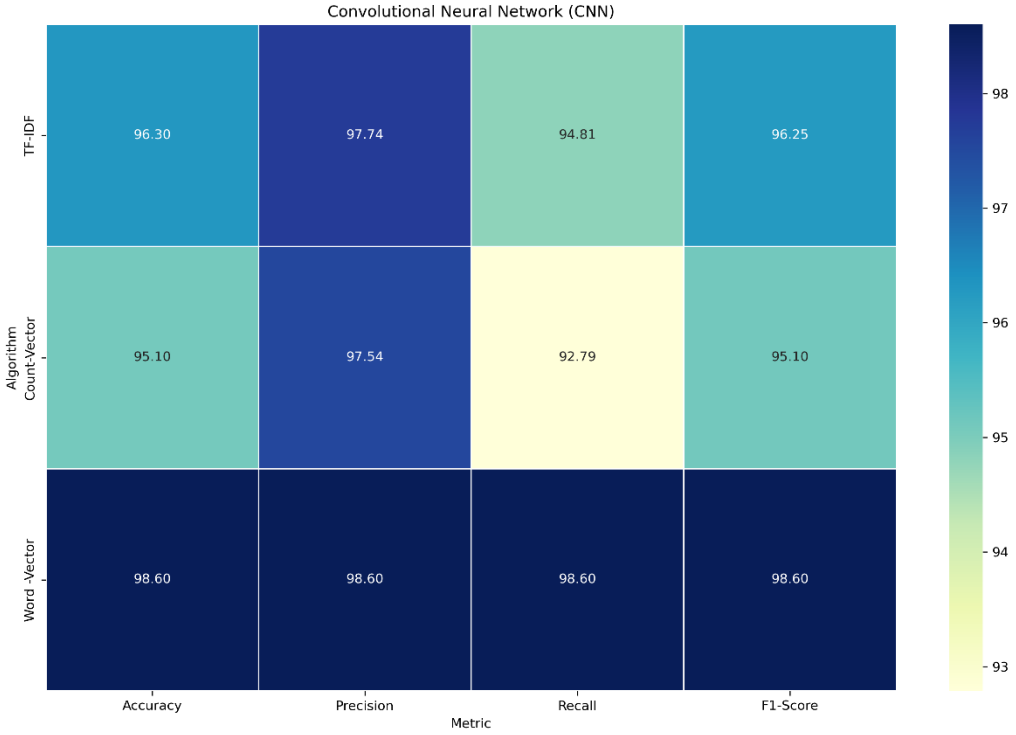


Figure 4.40 : Performance Metrics per CNN per Dataset2

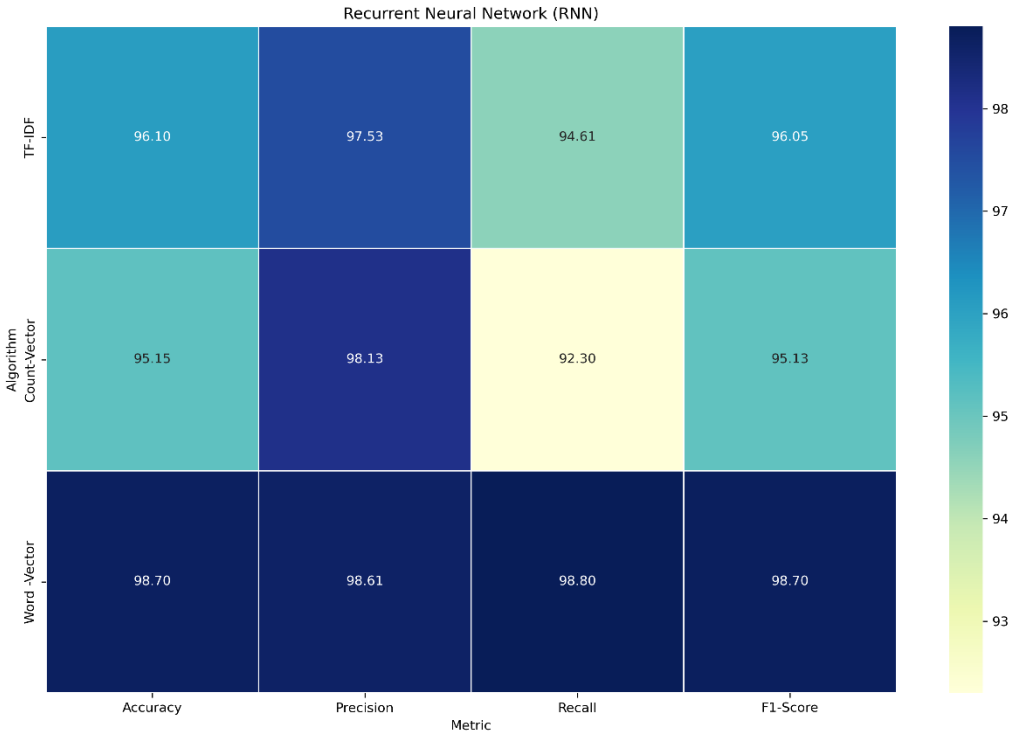


Figure 4.41 : Performance Metrics per RNN per Dataset2

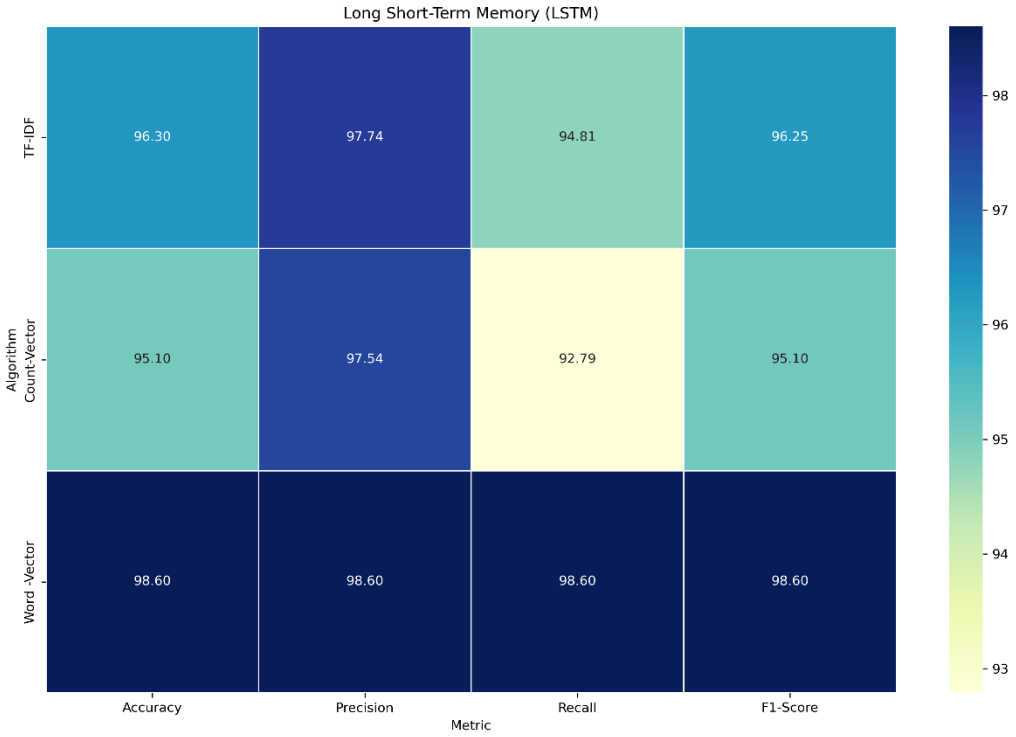


Figure 4.42 : Performance Metrics per LSTM per Dataset2

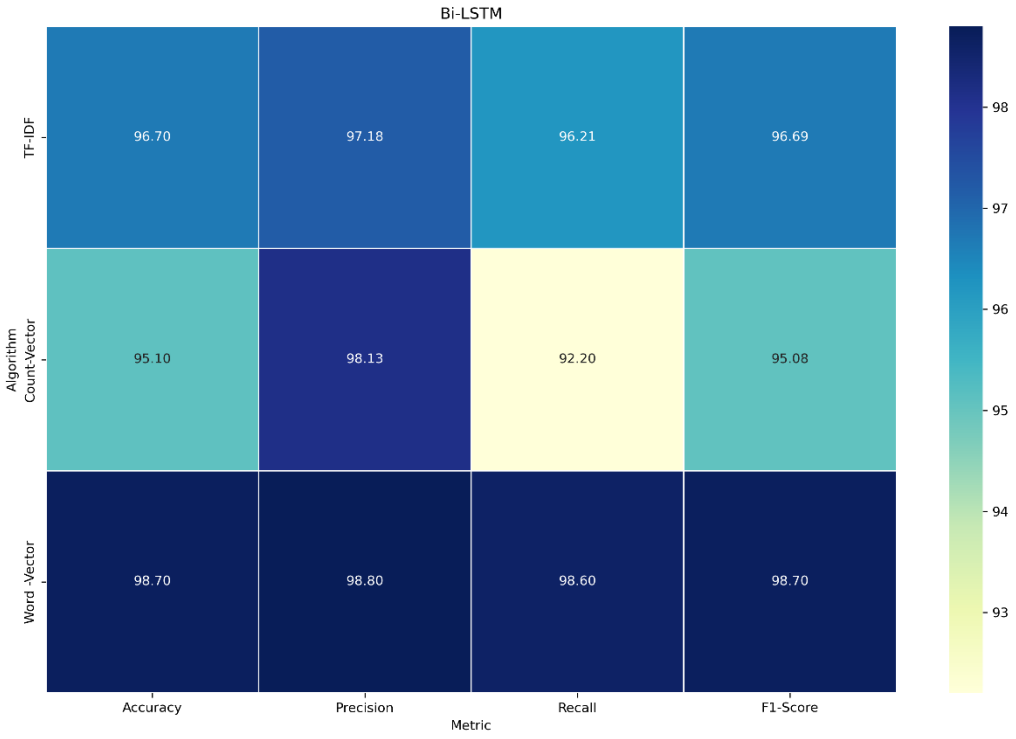


Figure 4.43 : Performance Metrics per Bi-LSTM per Dataset2

- The third group Figures shows the Performance Metrics of the hybrid Algorithms:

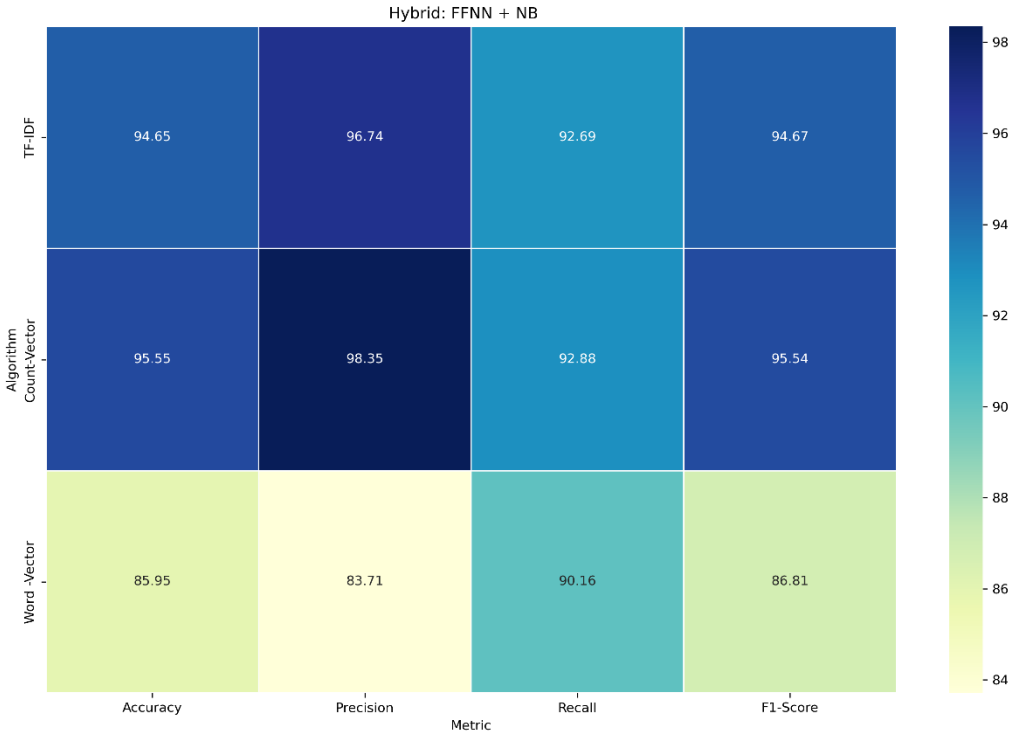


Figure 4.44 : Performance Metrics per Hybrid FFNN+NB per Dataset2

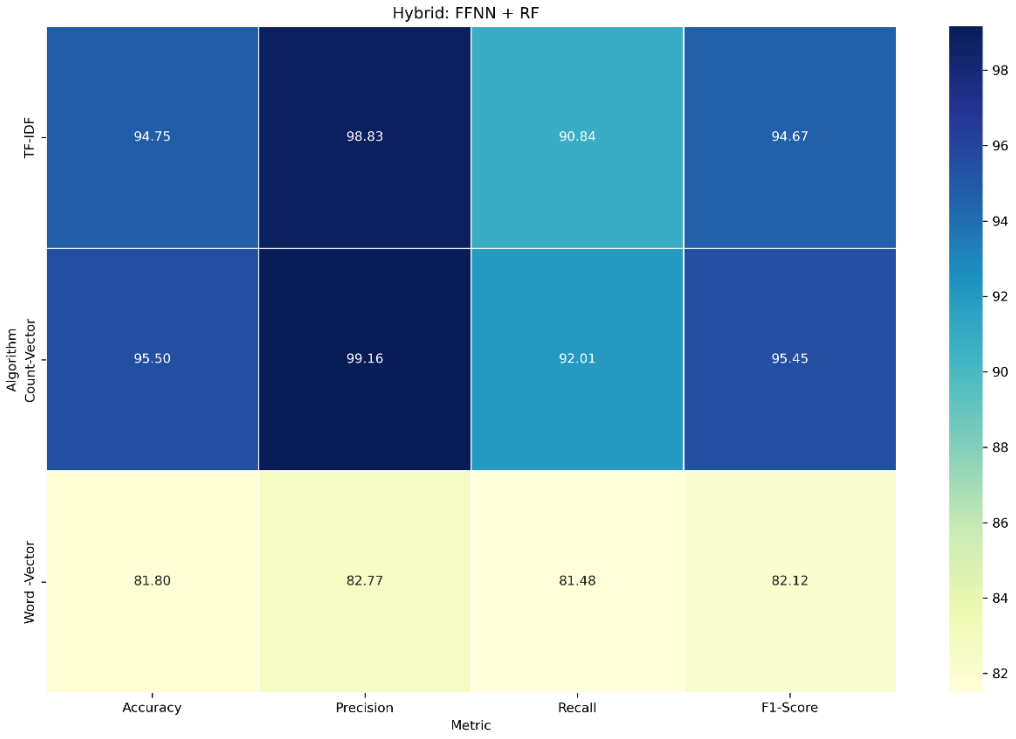


Figure 4.45 : Performance Metrics per Hybrid FFNN+RF per Dataset2

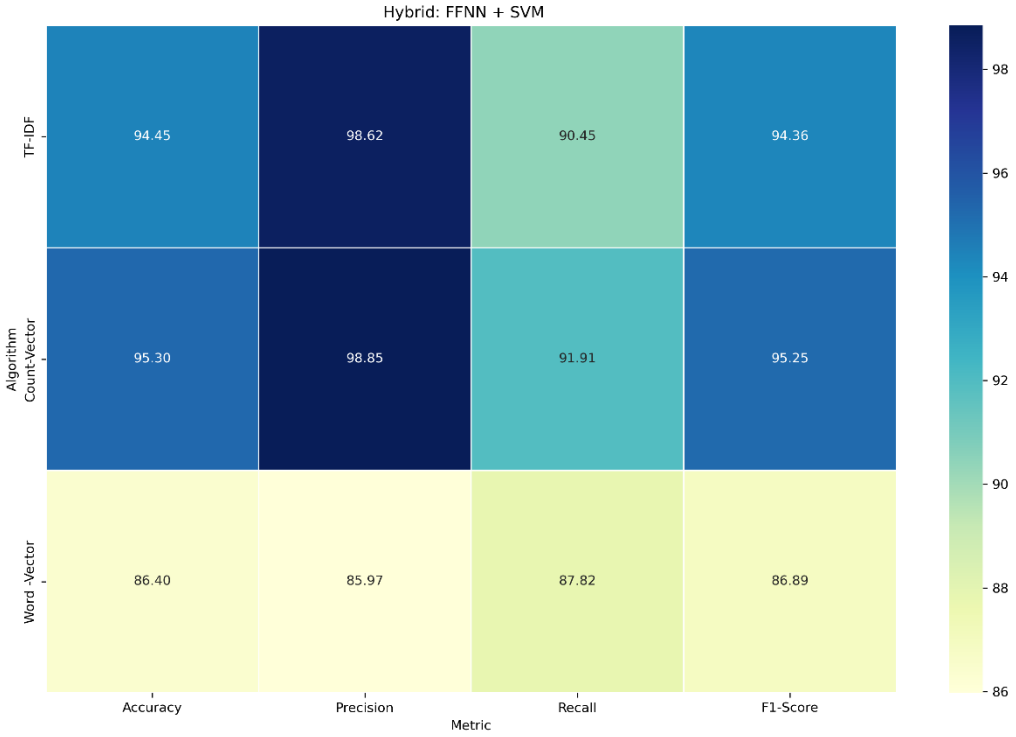


Figure 4.46 : Performance Metrics per Hybrid FFNN+SVM per Dataset2

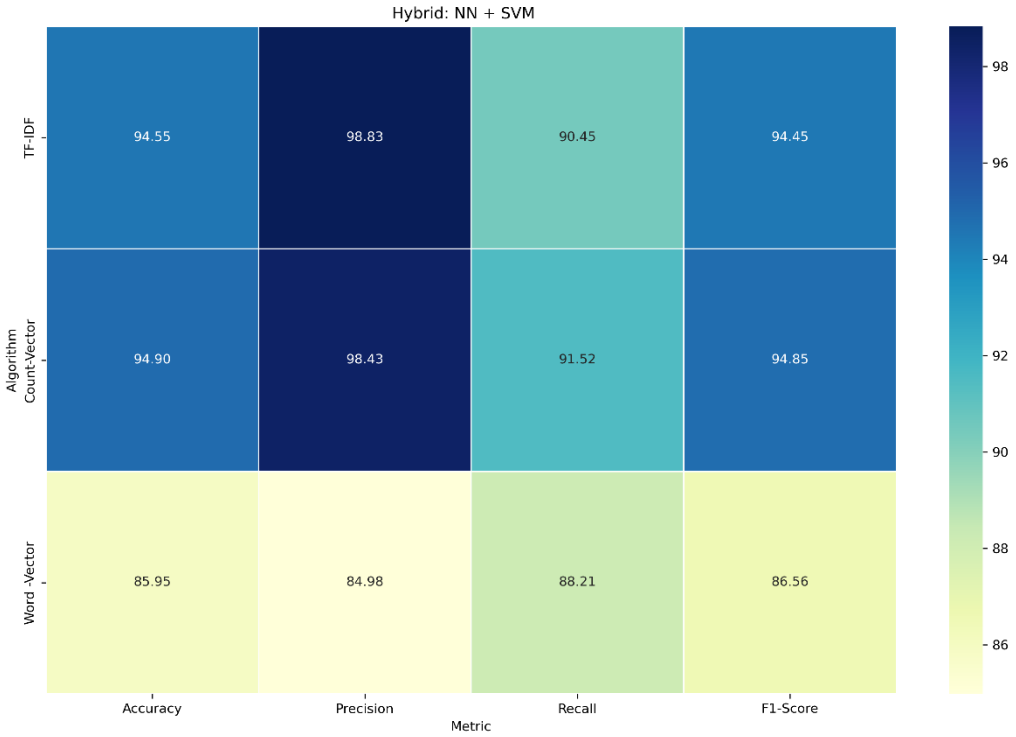


Figure 4.47 : Performance Metrics per Hybrid NN+SVM per Dataset2

4.2.3.2 Execution Time Results

This section displays the actual execution times for each dataset independently. Two Figures, one for the first dataset and one for the second, compile and display the time differences between using Count Vectorizer and Word Vectorizer and TF-IDF.

-The following group of Figures shows the actual execution time difference for Dataset 1:

1- Traditional Algorithms:

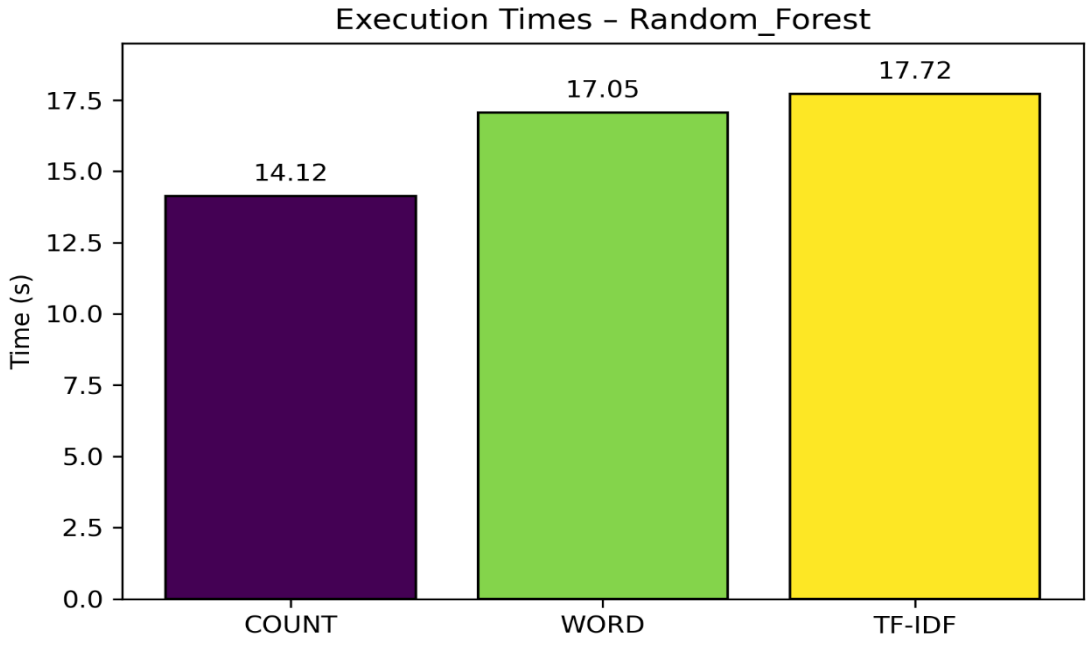


Figure 4.48 : time execution of Random Forest of Dataset1

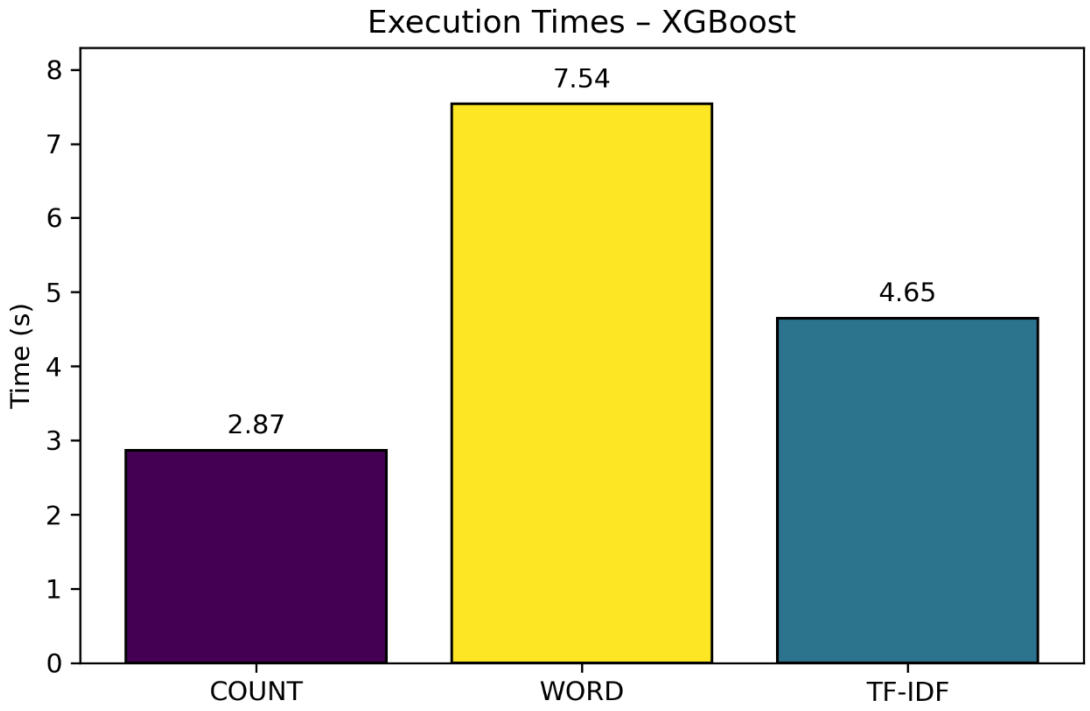


Figure 4.49 : time execution of XGBOOST of Dataset1

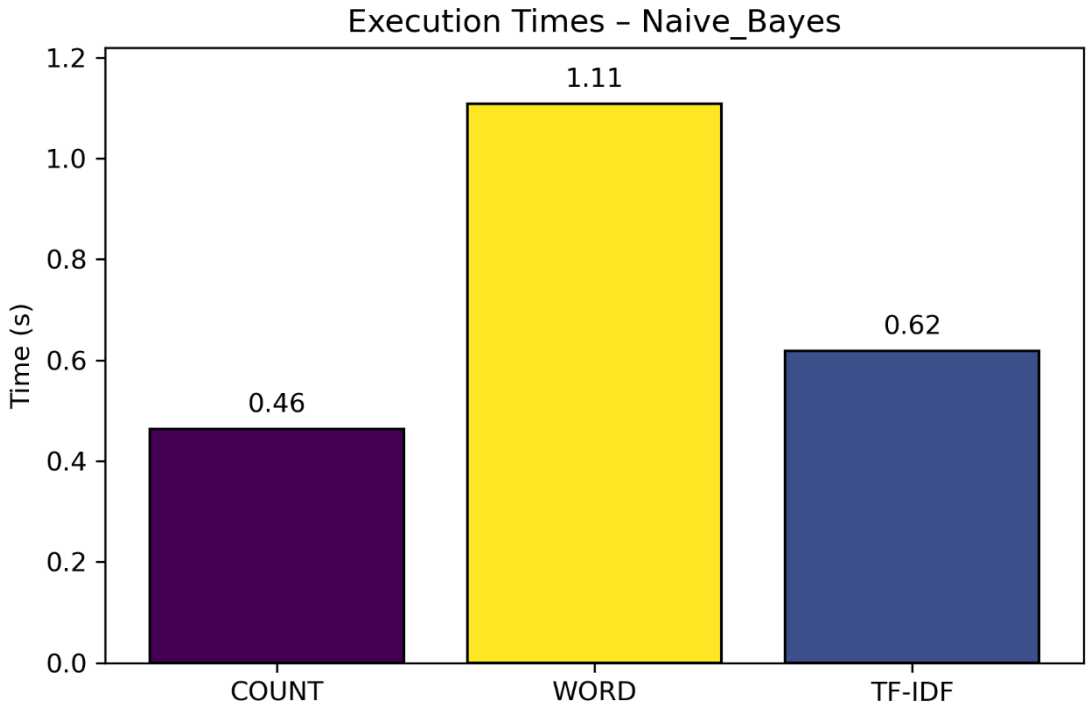


Figure 4.50 : time execution of Naive Bayes of Dataset1

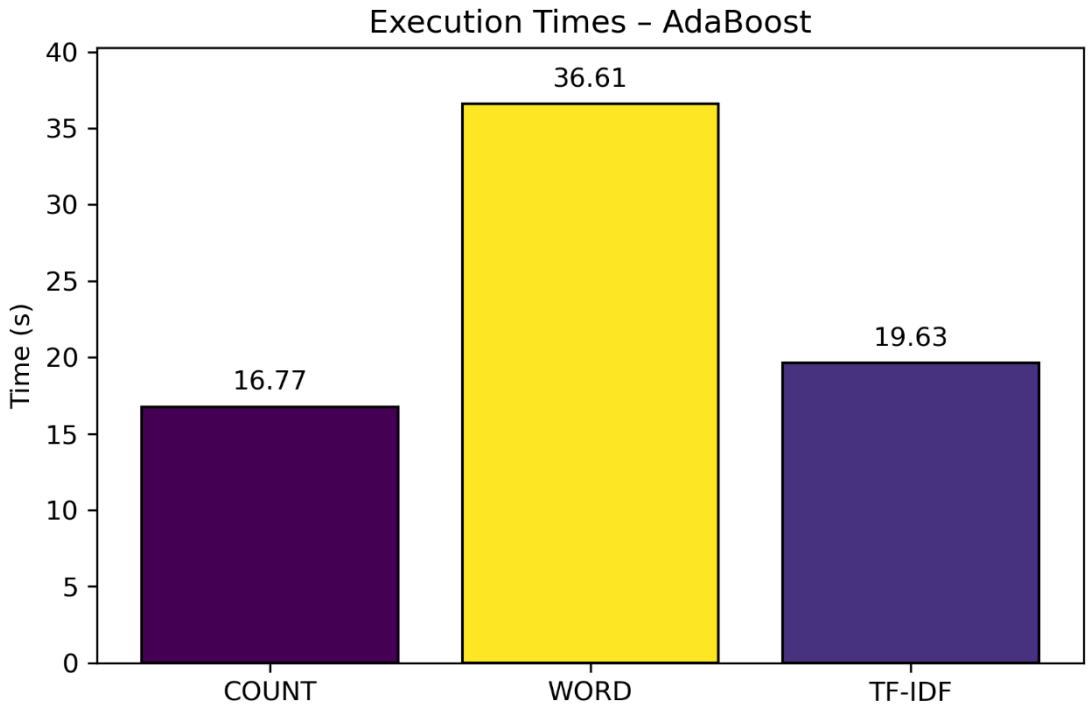


Figure 4.51 : time execution of AdaBoost of Dataset1

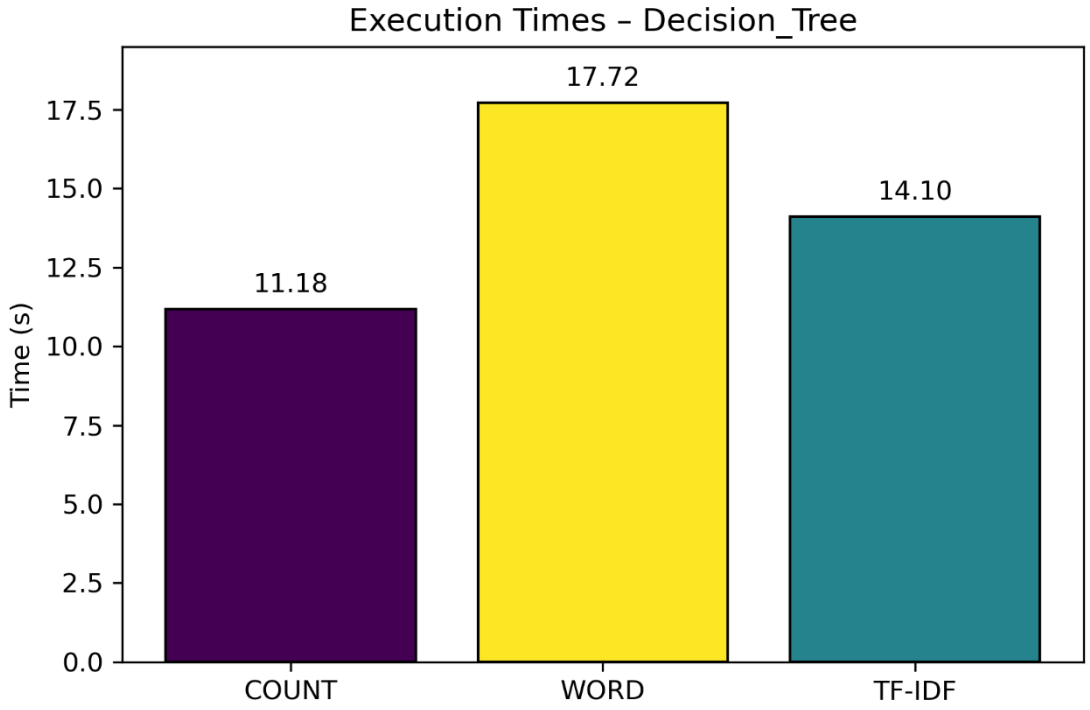


Figure 4.52 : time execution of Decision Tree of Dataset1

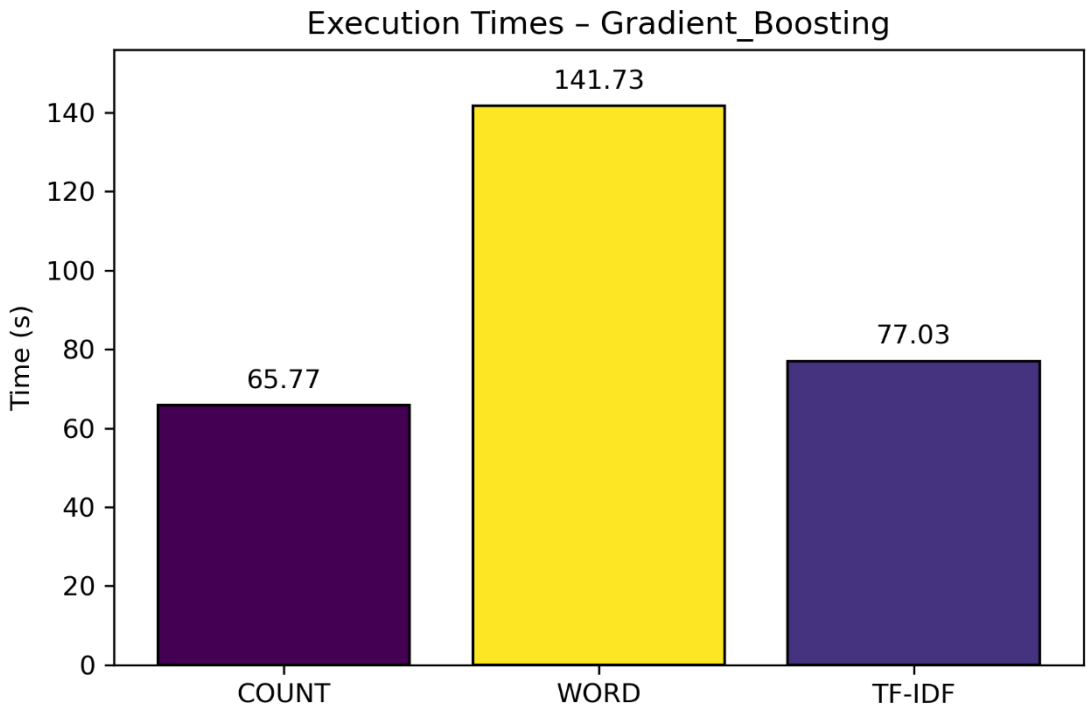


Figure 4.53 : time execution of Gradient Boosting of Dataset1

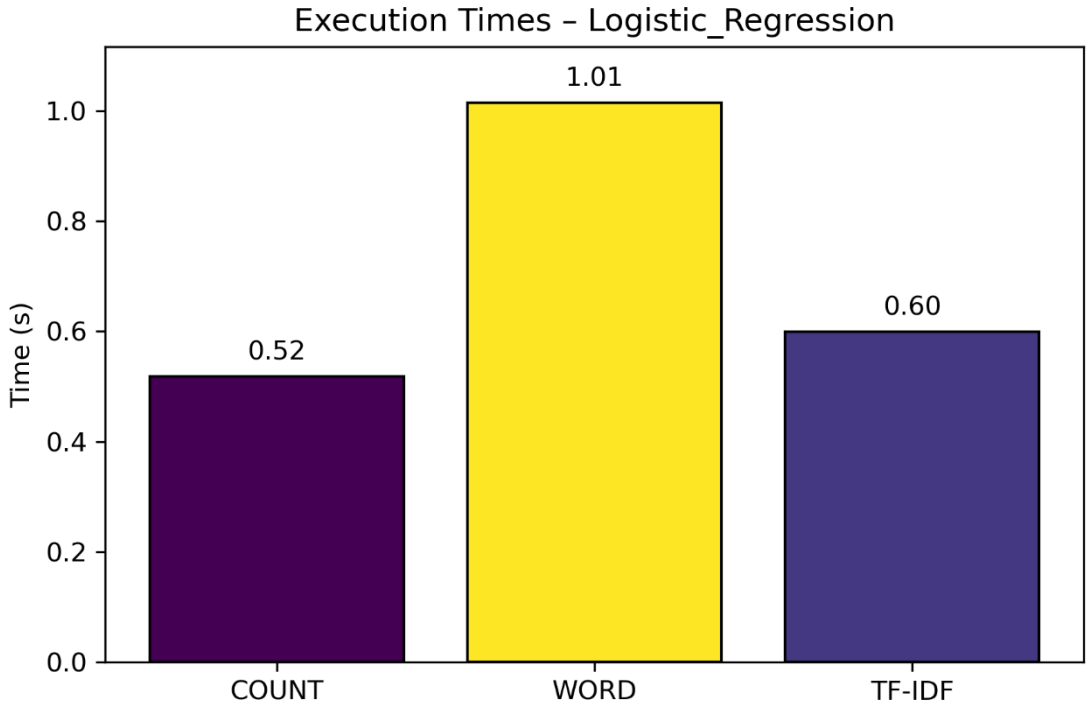


Figure 4.54 : time execution of Logistic Regression of Dataset1

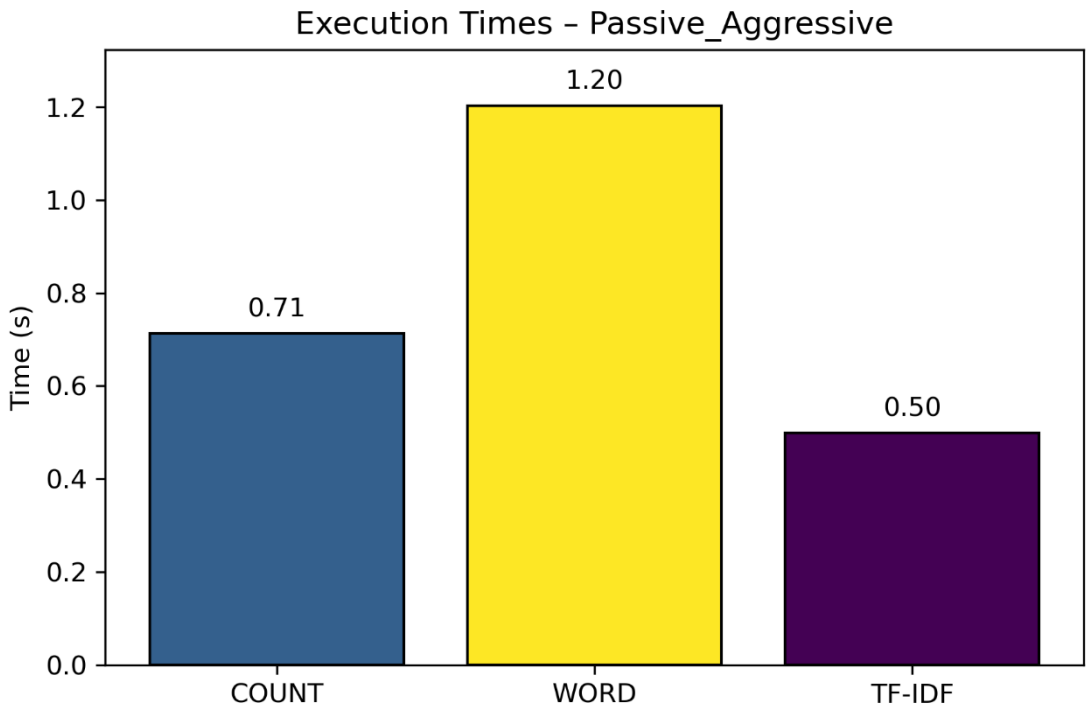


Figure 4.55 : time execution of Passive Aggressive of Dataset1

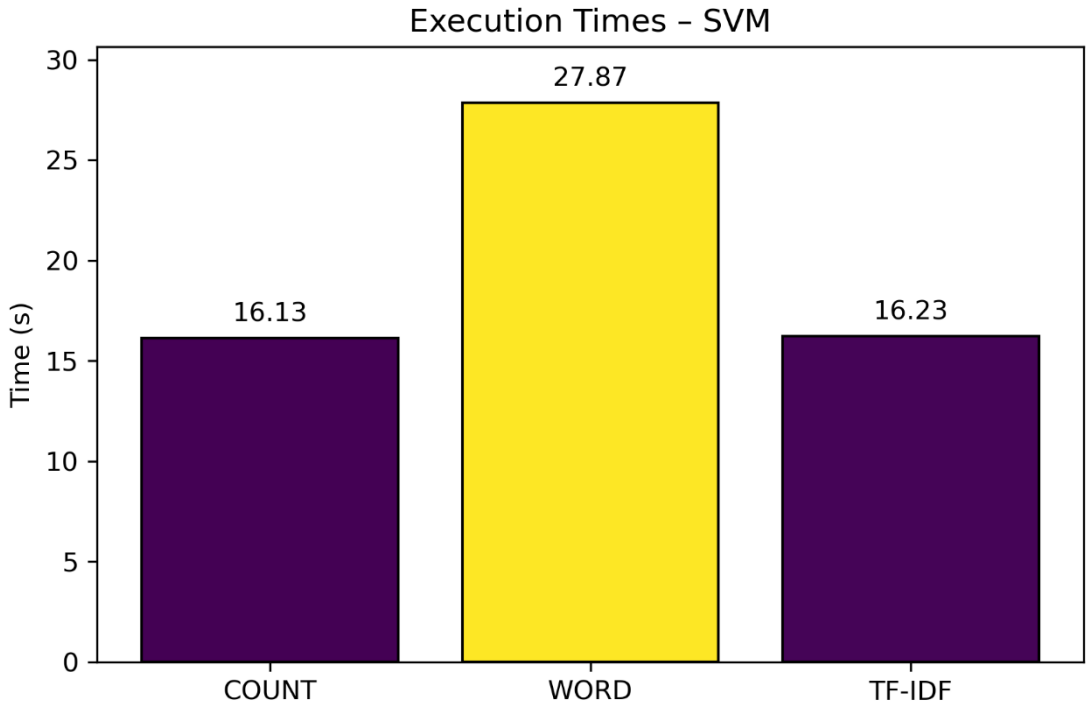


Figure 4.56 : time execution of SVM of Dataset1

2-Deep learning Algorithms:

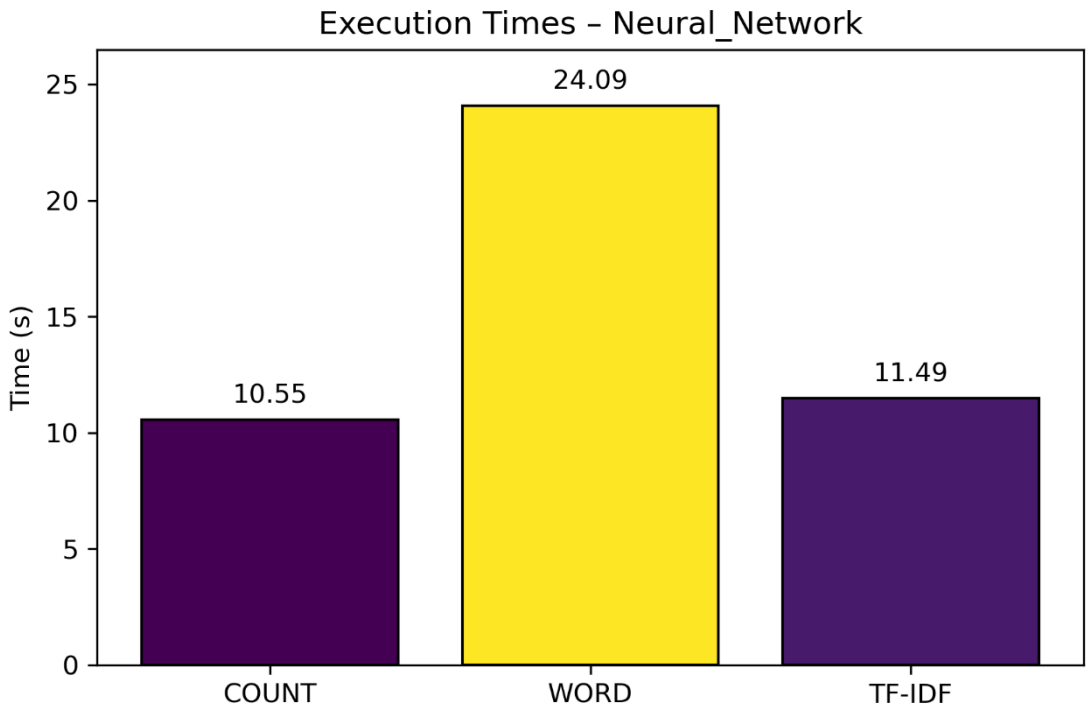


Figure 4.57 : time execution of Neural Network of Dataset1

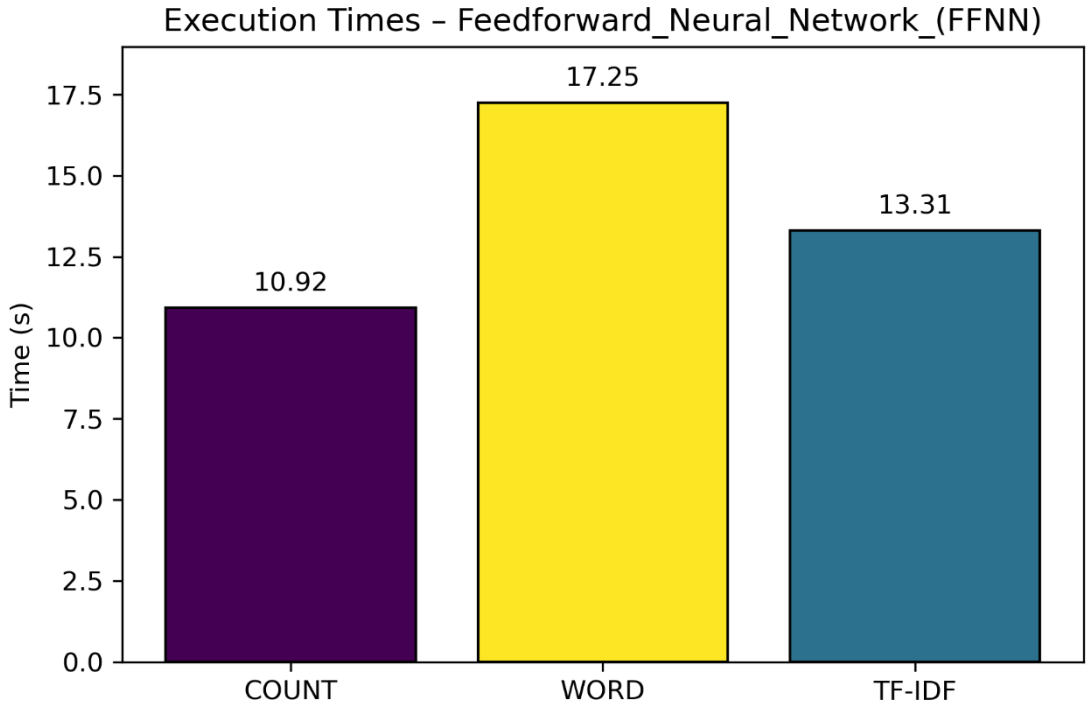


Figure 4.58 : time execution of FFNN of Dataset1

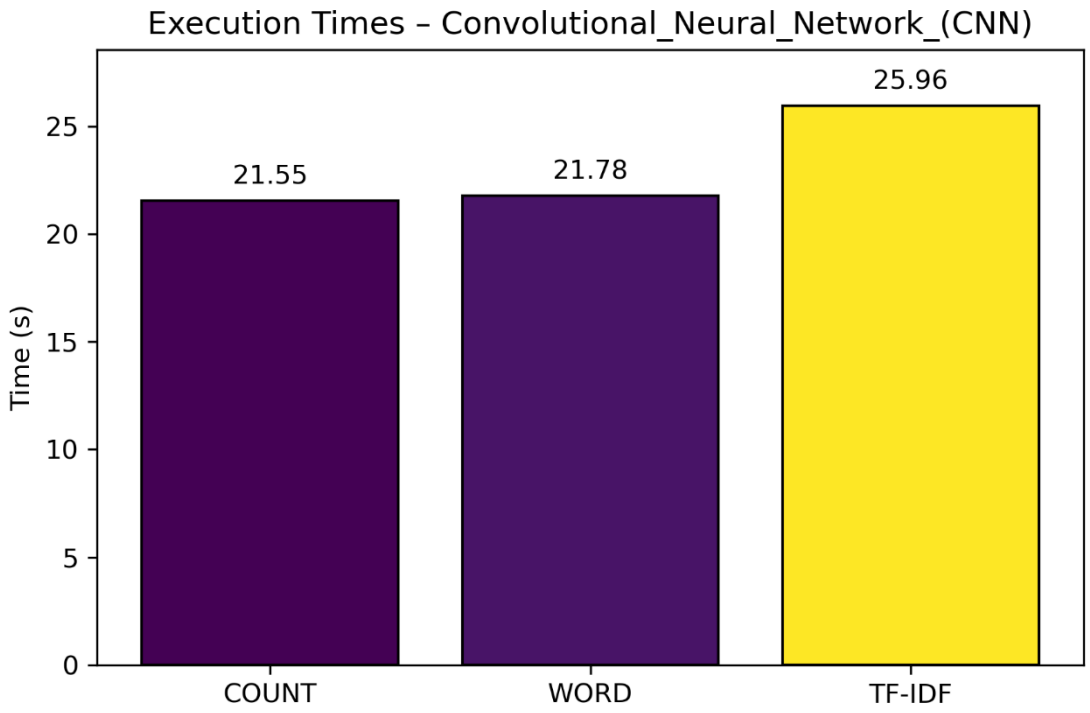


Figure 4.59 : time execution of CNN of Dataset1

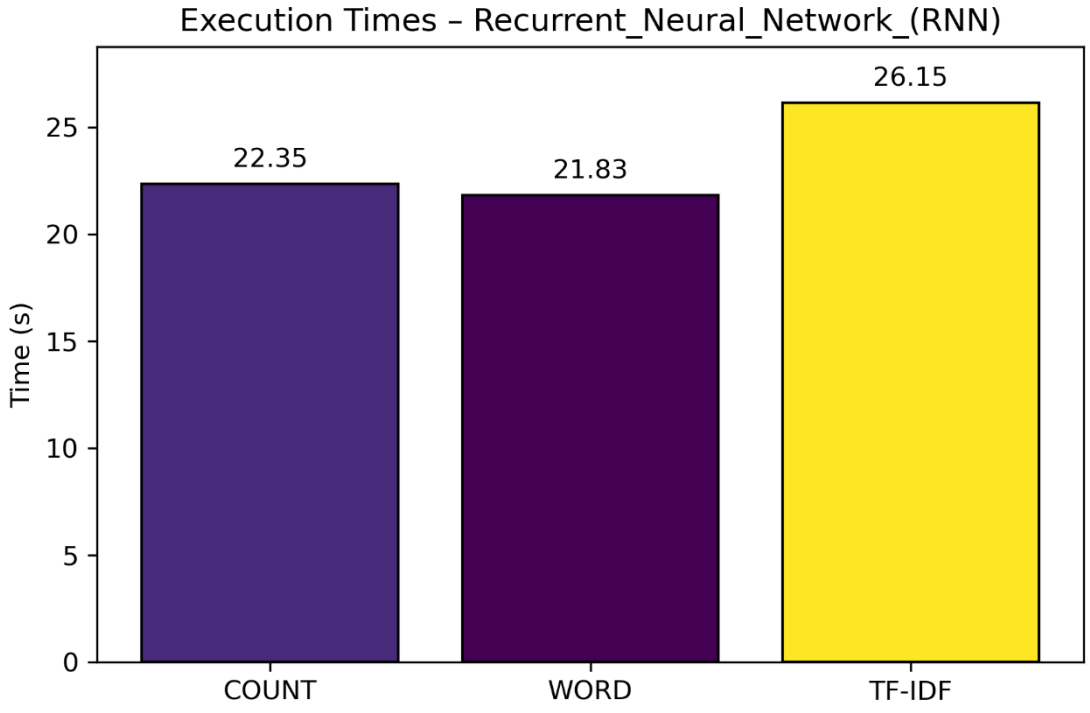


Figure 4.60 : time execution of RNN of Dataset1

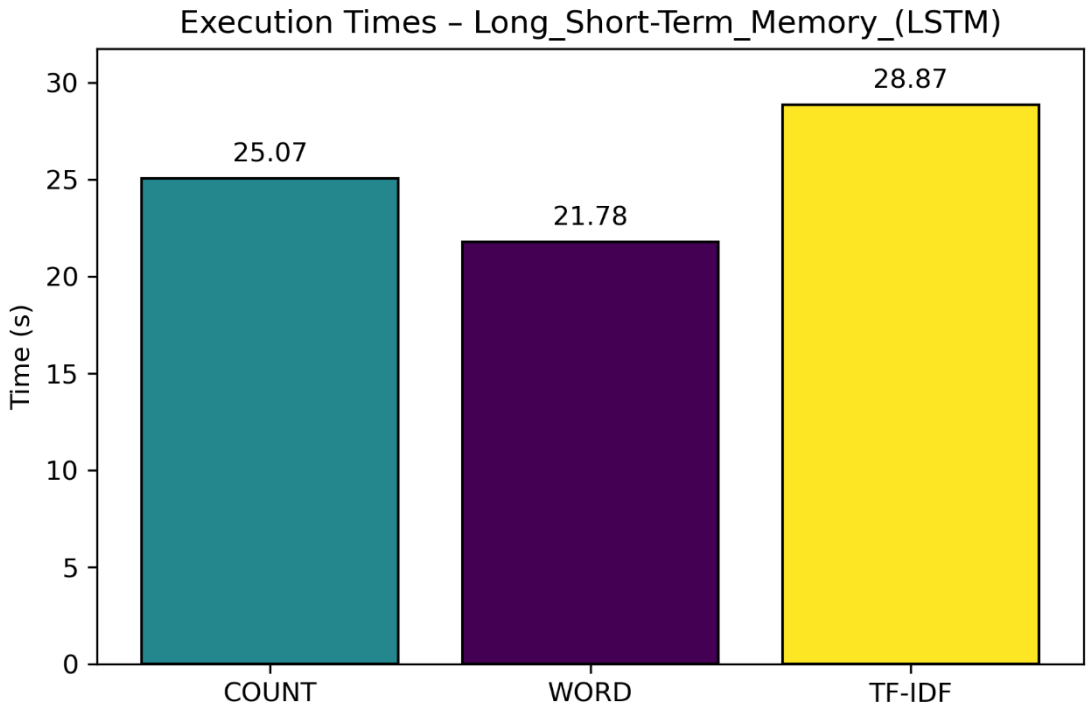


Figure 4.61 : time execution of LSTM of Dataset1

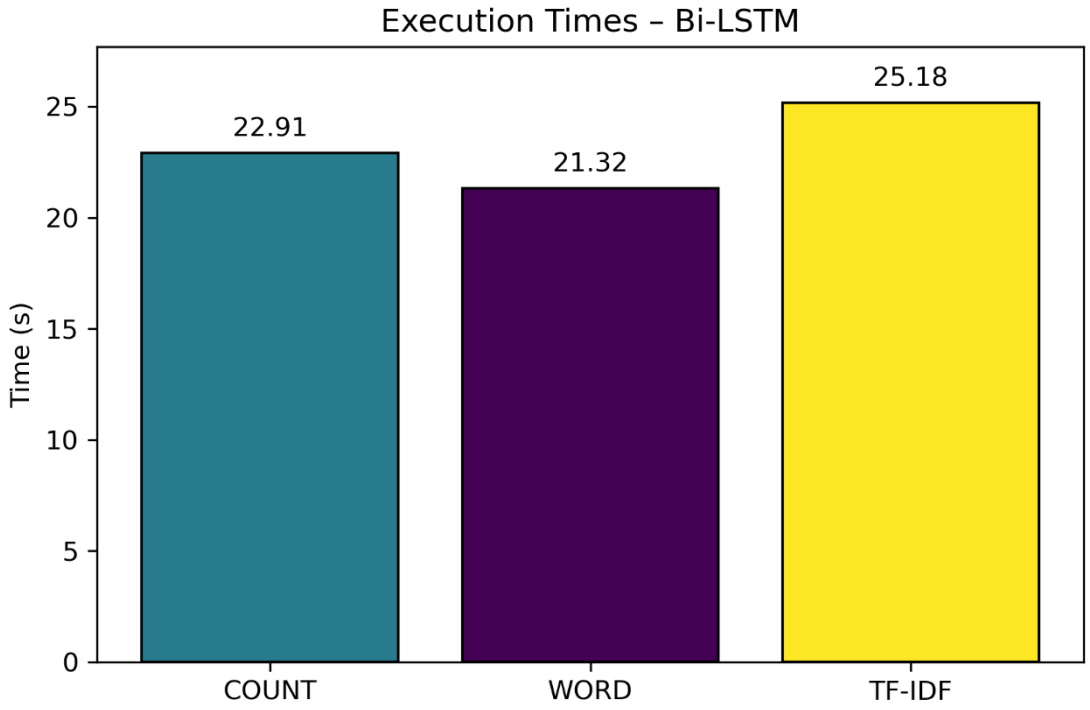


Figure 4.62 : time execution of Bi-LSTM of Dataset1

3-Hybrid Algorithms:

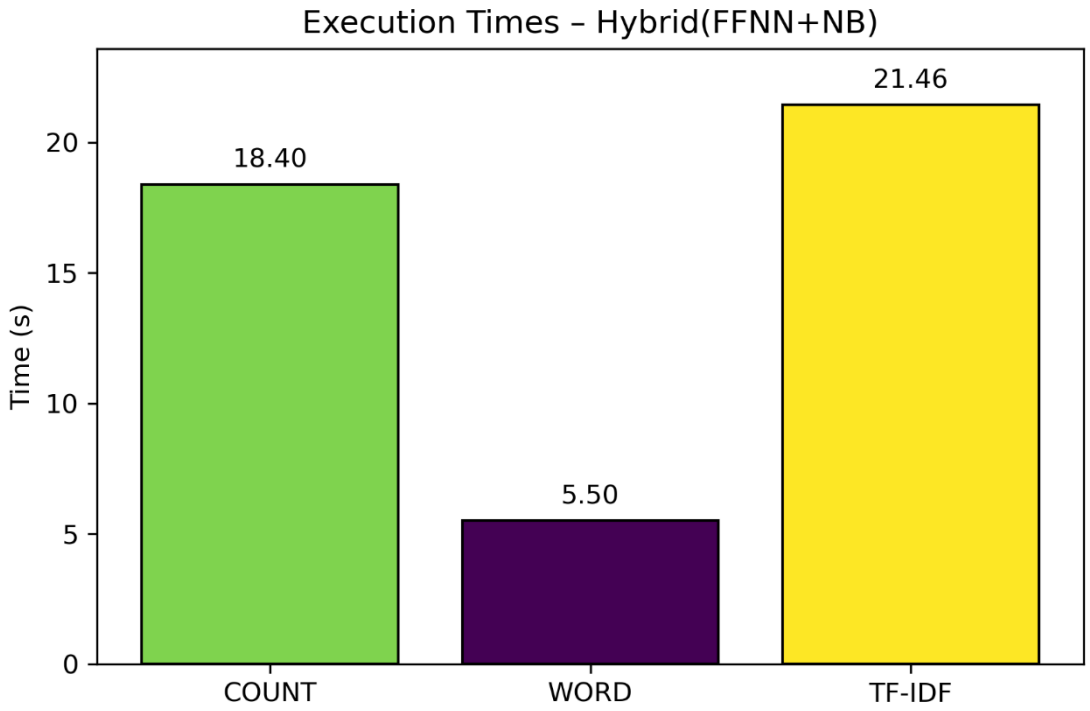


Figure 4.63 : Time execution of Hybrid FFNN+NB of Dataset1

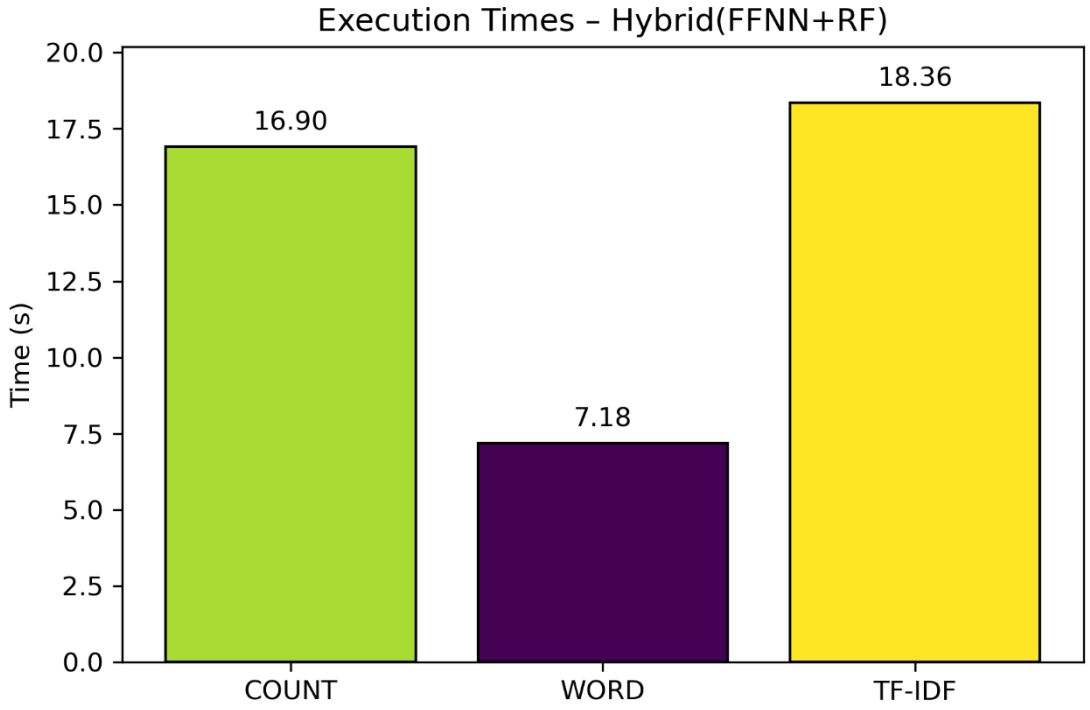


Figure 4.64 : time execution of Hybrid FFNN+RF of Dataset1

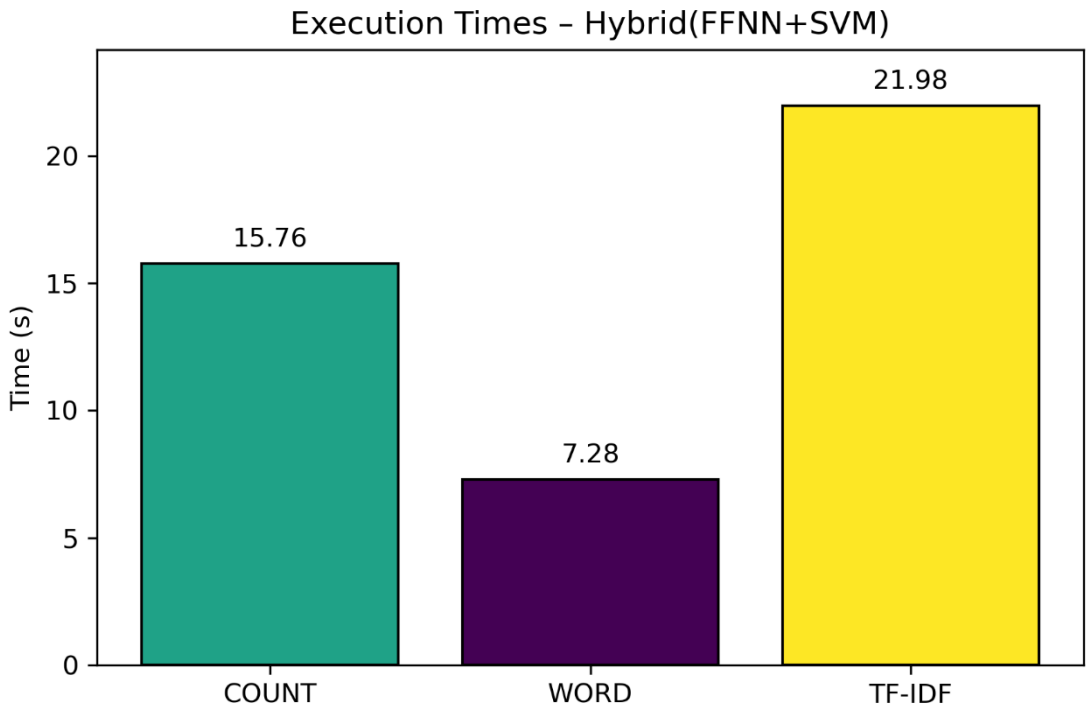


Figure 4.65 : Time execution of Hybrid FFNN+SVM of Dataset1

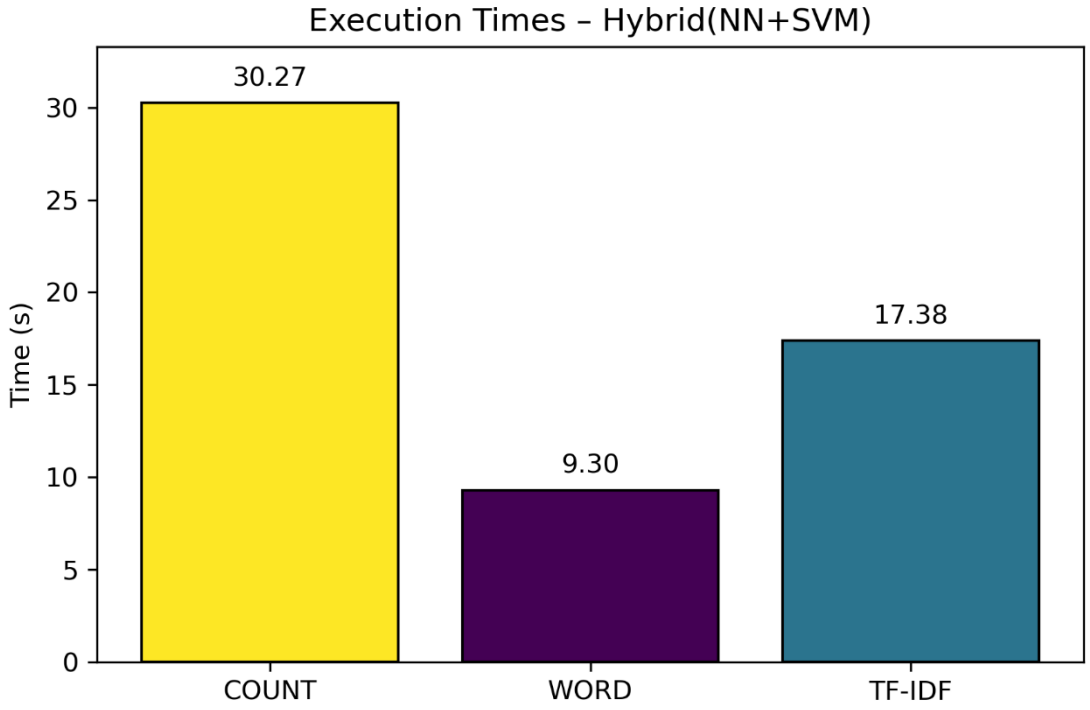


Figure 4.66 : Time execution of Hybrid NN+SVM of Dataset1

-The following group of Figures shows the actual execution time difference for Dataset 2:

1-Traditional Algorithms:

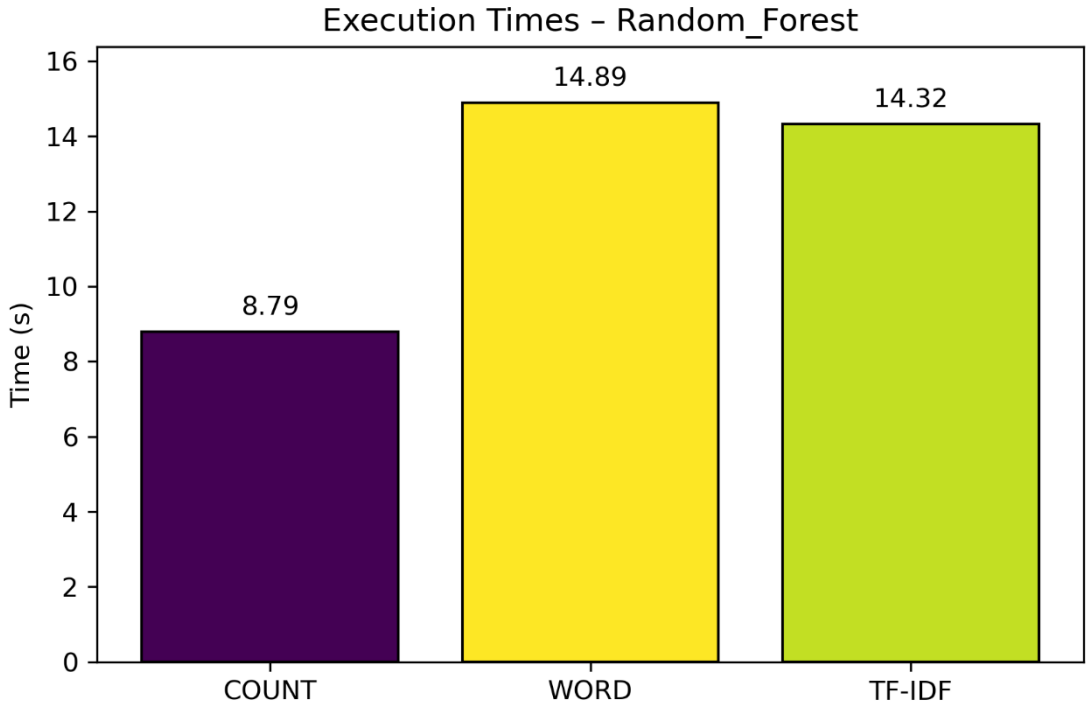


Figure 4.67 : Time execution of Random Forest of Dataset2

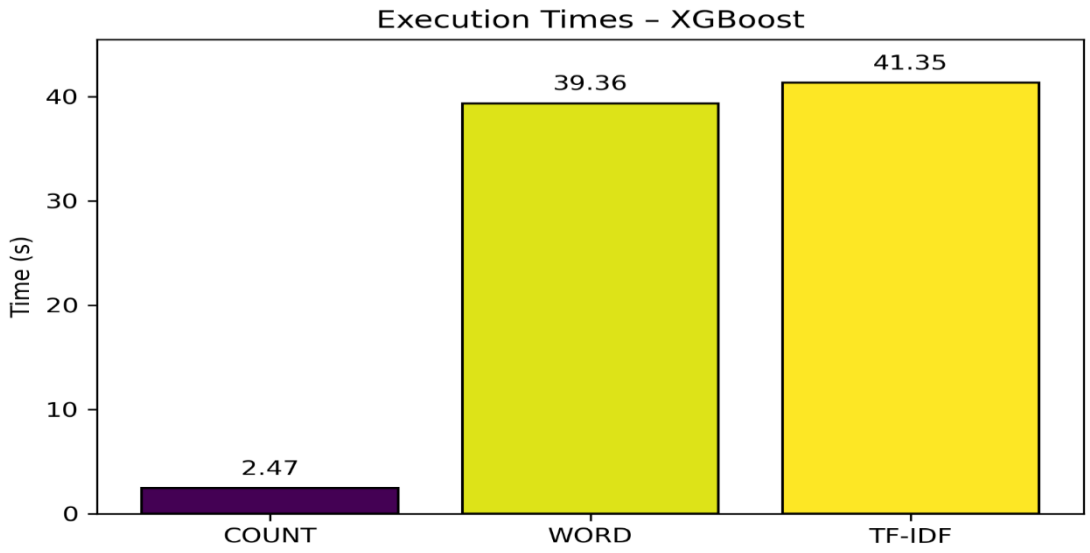


Figure 4.68 : Time execution of XGBoost of Dataset2

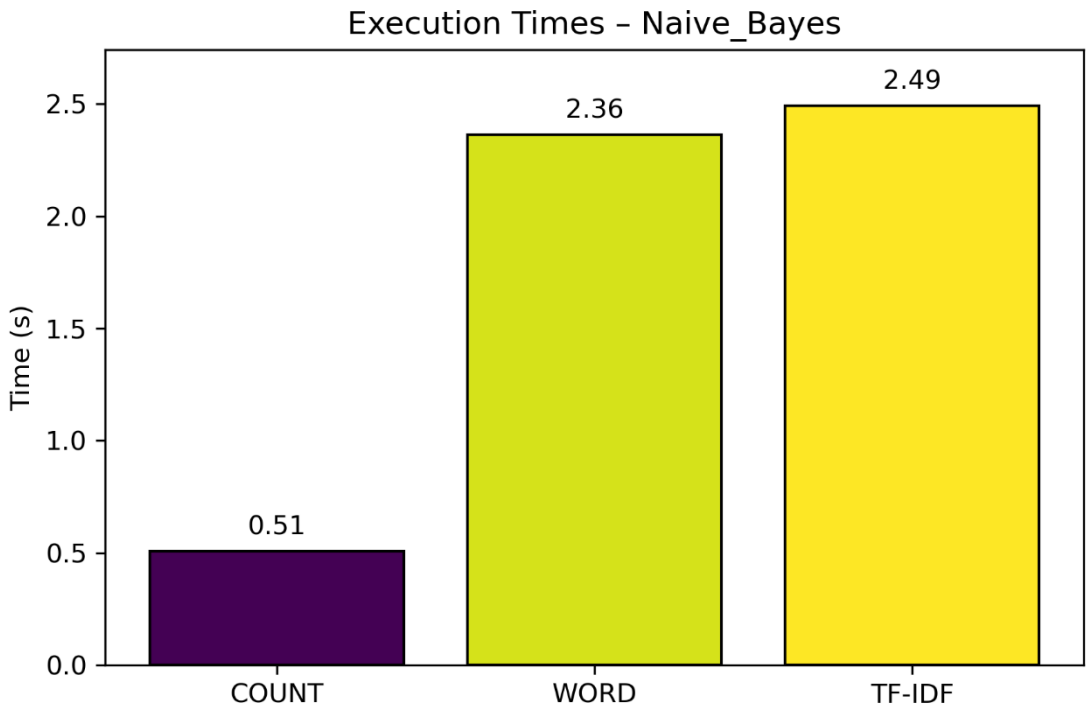


Figure 4.69 : Time execution of Naive Bayes of Dataset2

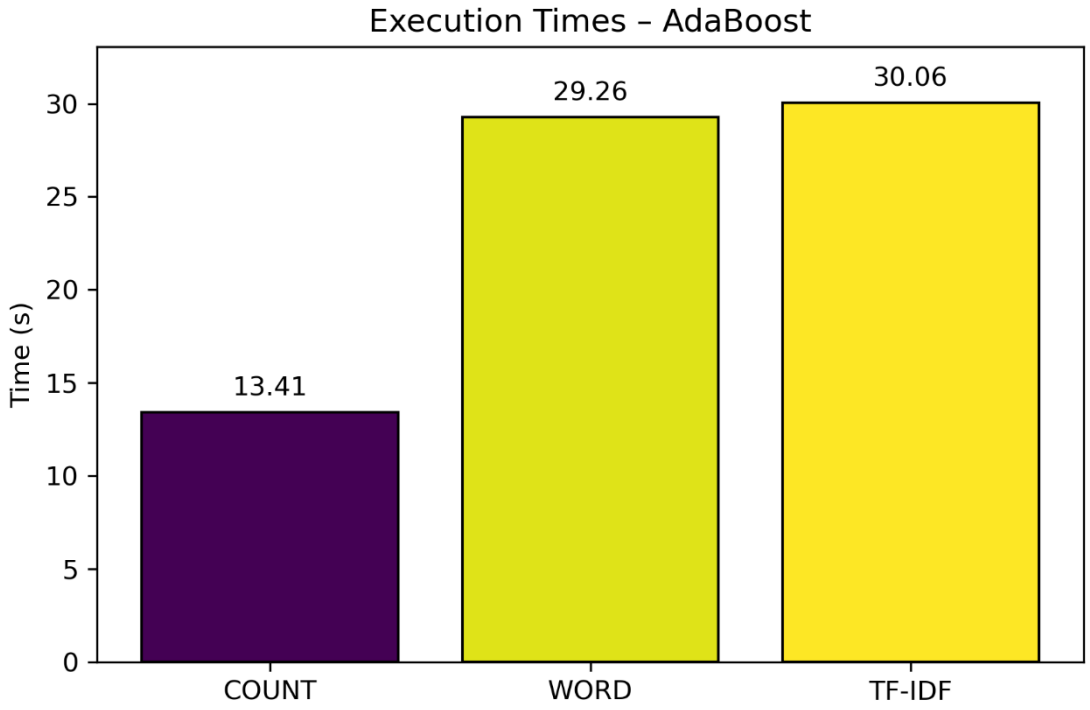


Figure 4.70 : Time execution of AdaBoost of Dataset2

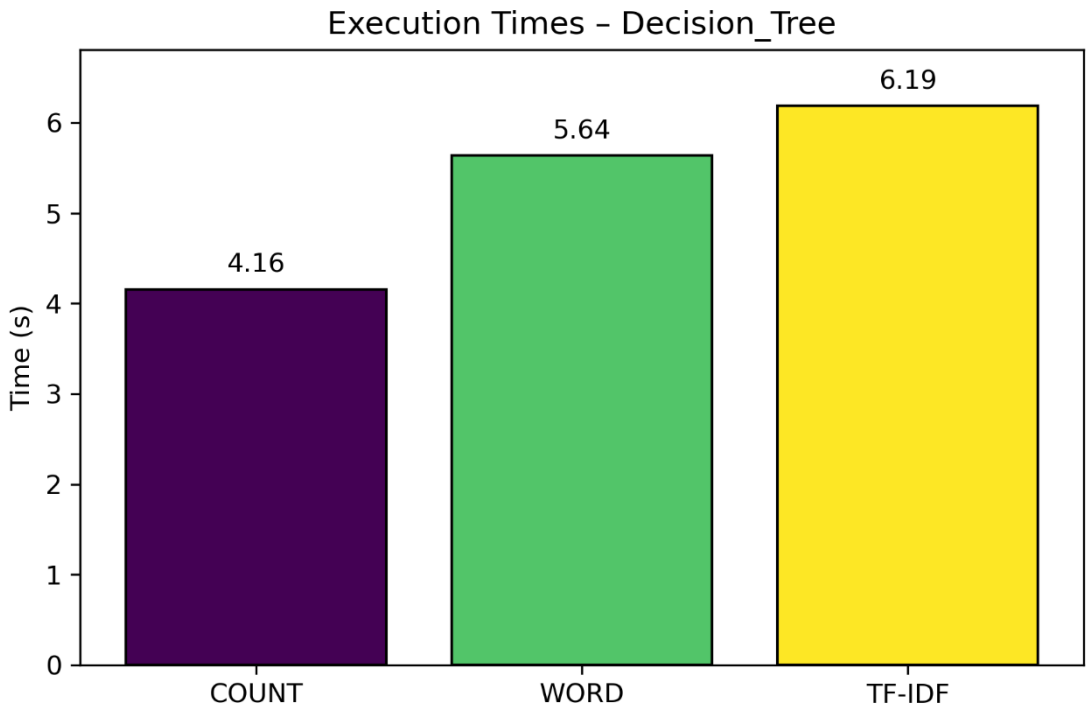


Figure 4.71 : Time execution of Decision Tree of Dataset2

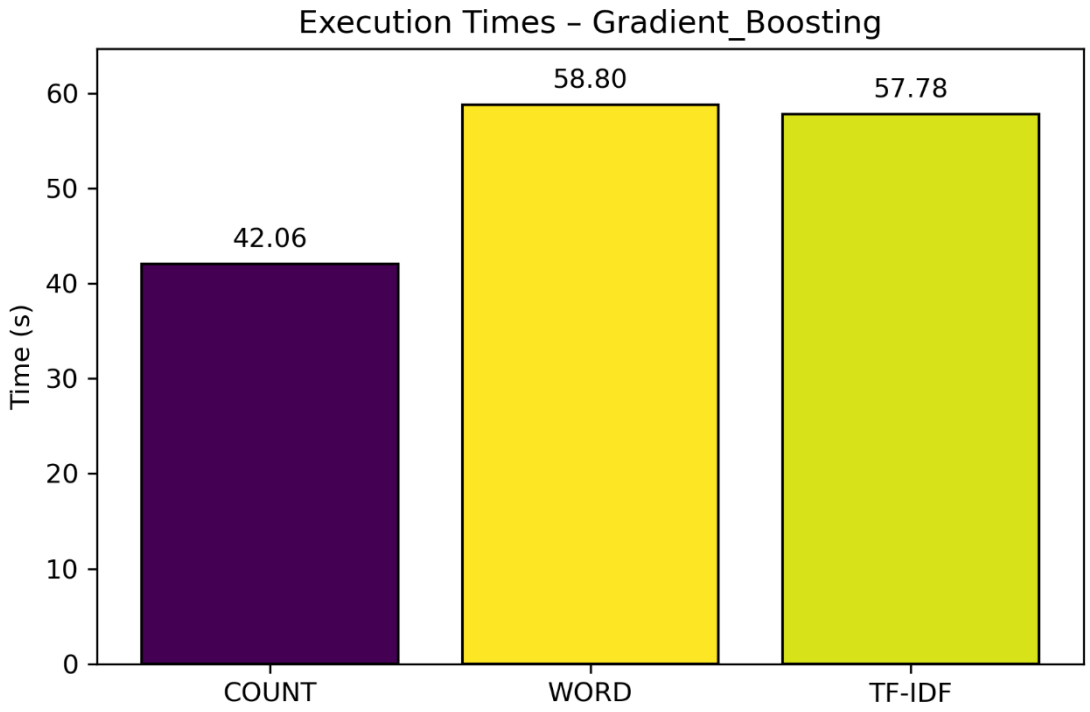


Figure 4.72 : Time execution of Gradient Boosting of Dataset2

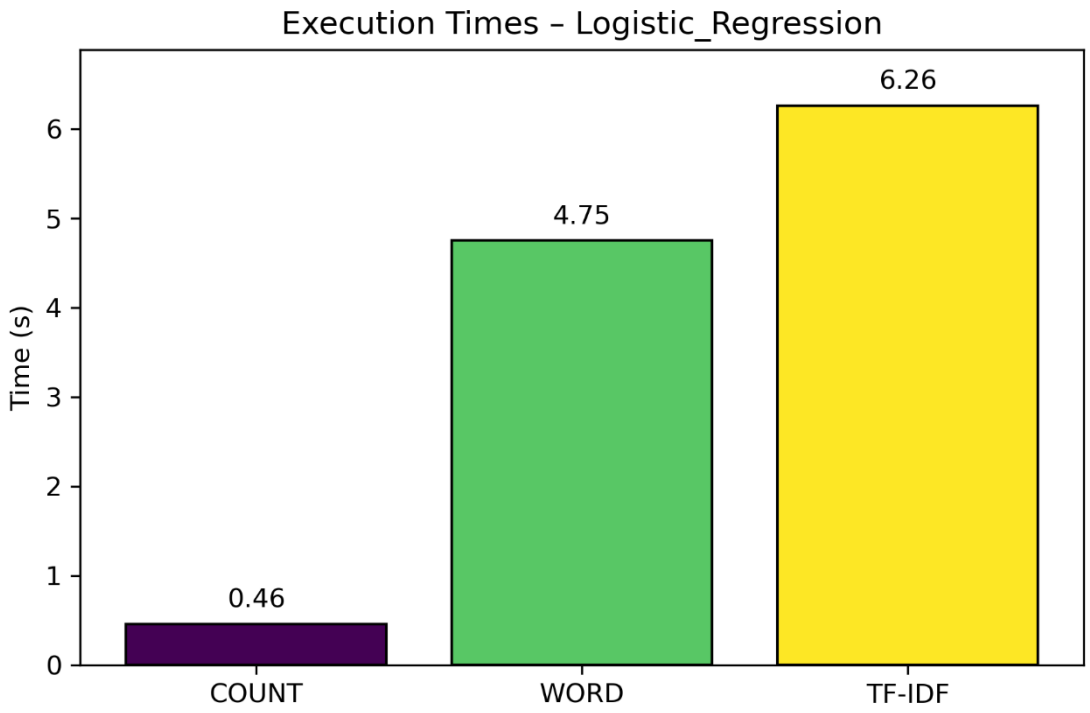


Figure 4.73 : Time execution of Logistic Regression of Dataset2

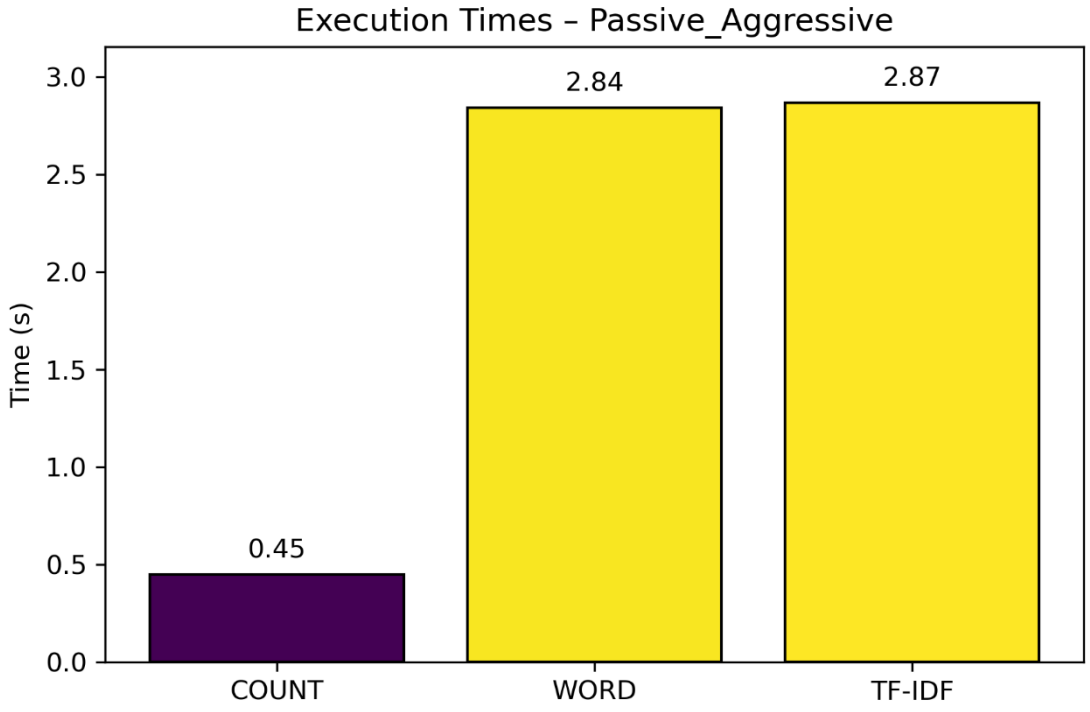


Figure 4.74 : Time execution of Passive Aggressive of Dataset2

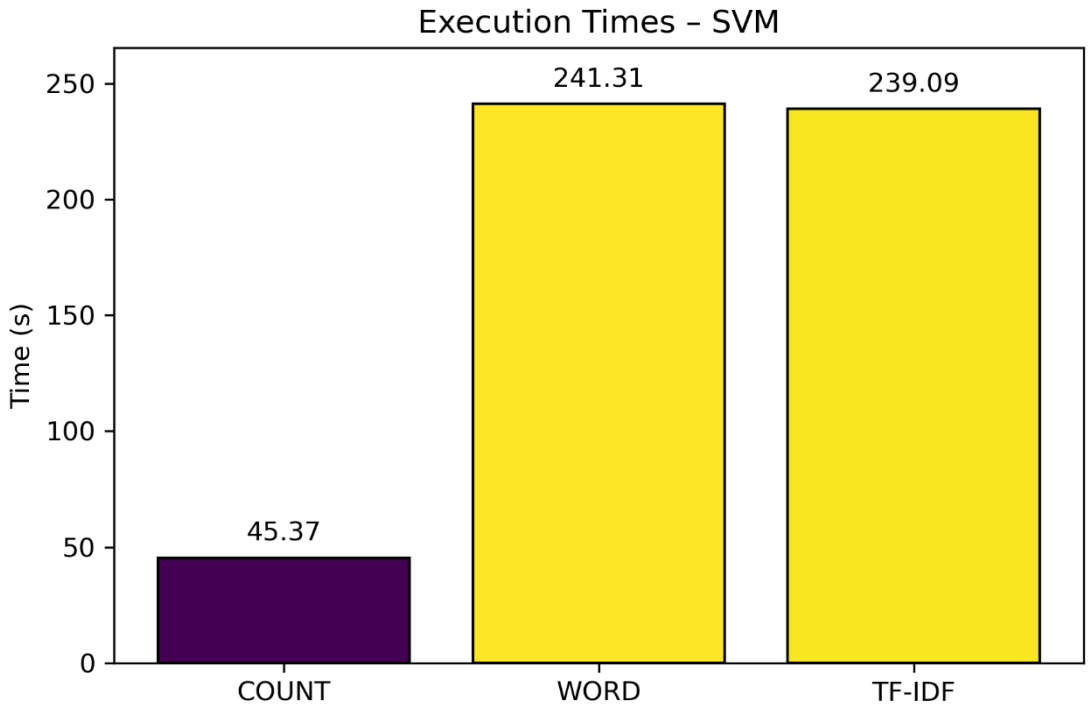


Figure 4.75 : Time execution of SVM of Dataset2

2-Deep learning Algorithms:

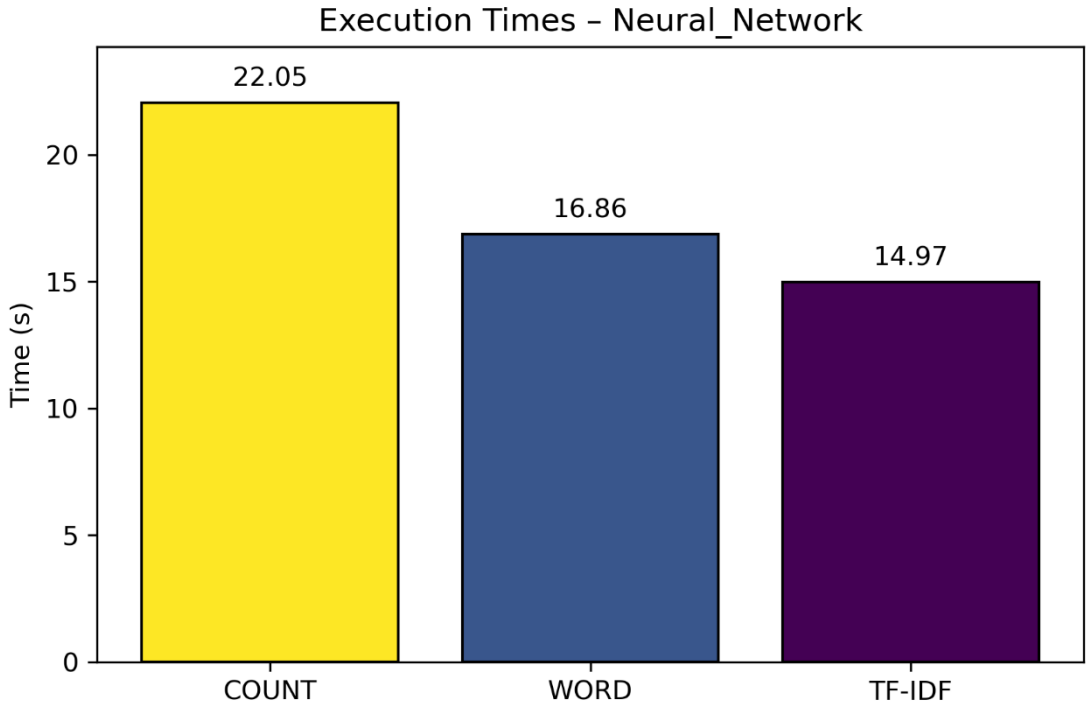


Figure 4.76 : Time execution of Neural Network of Dataset2

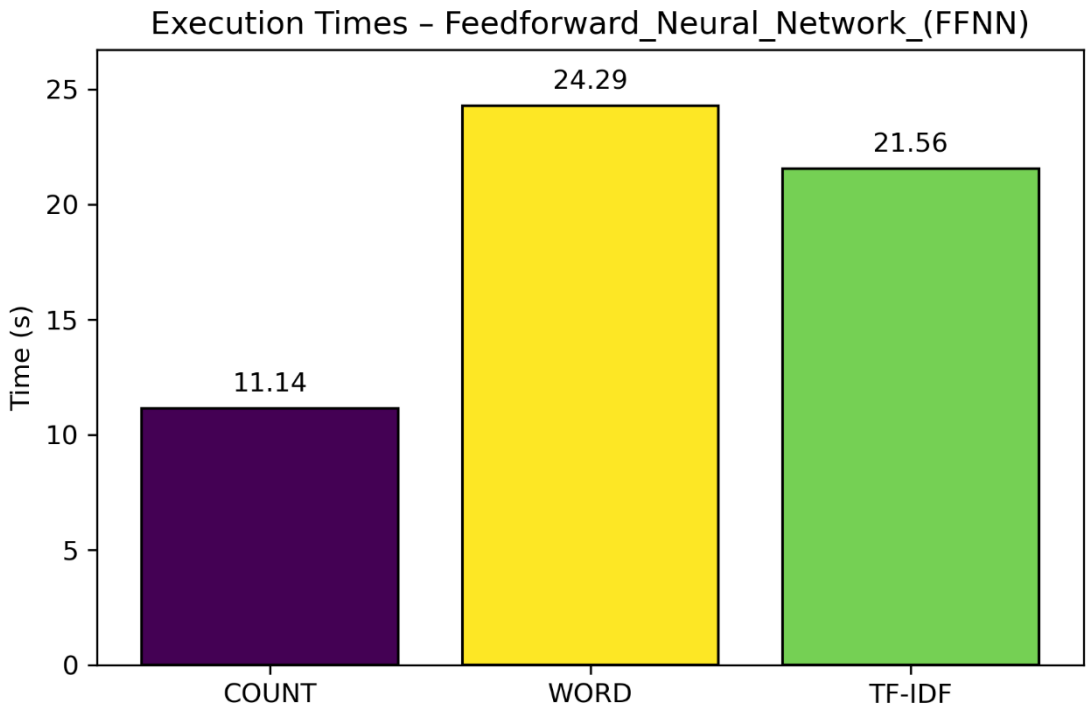


Figure 4.77 : Time execution of FFNN of Dataset2

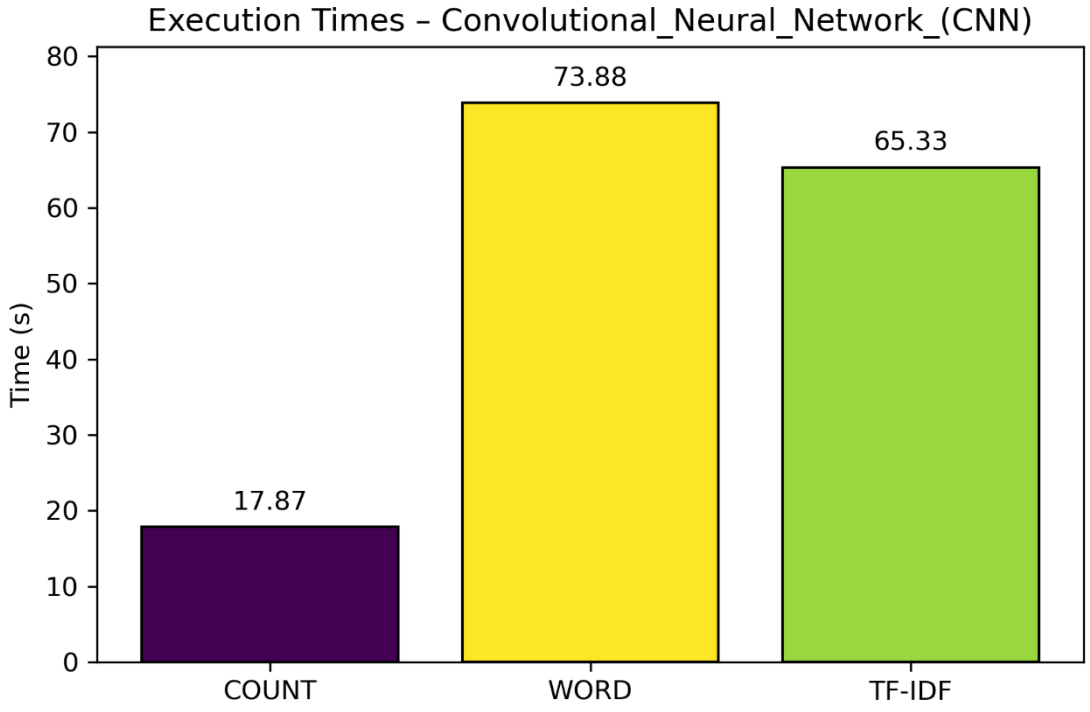


Figure 4.78 : Time execution of CNN of Dataset2

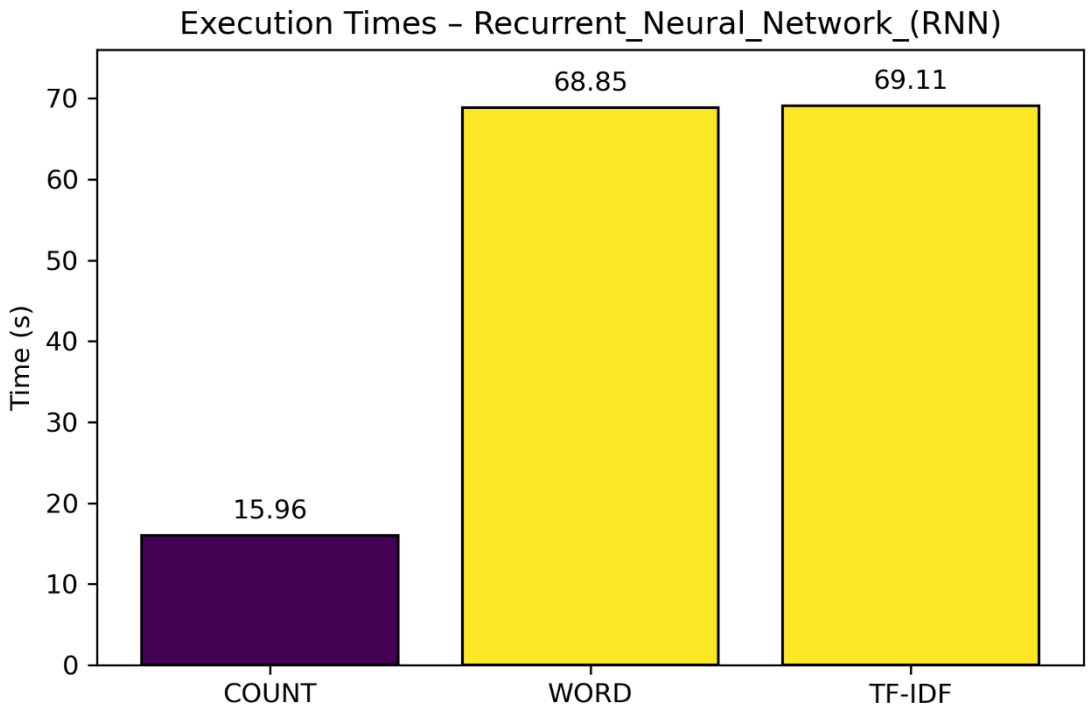


Figure 4.79 : Time execution of RNN of Dataset2

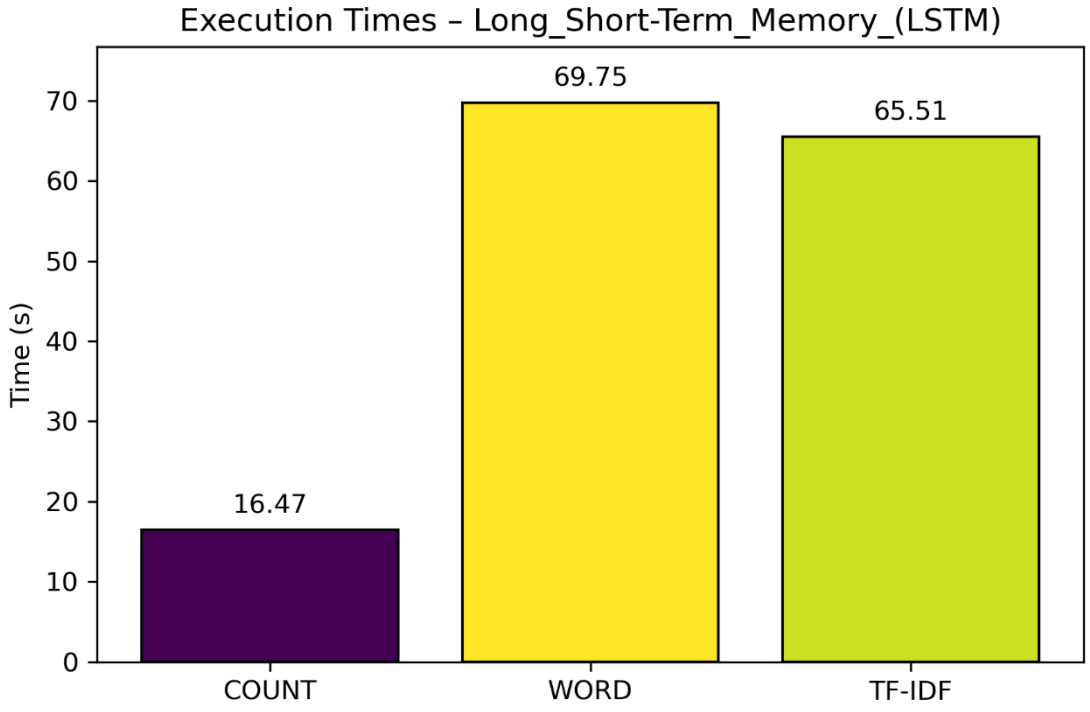


Figure 4.80 : Time execution of LSTM of Dataset2

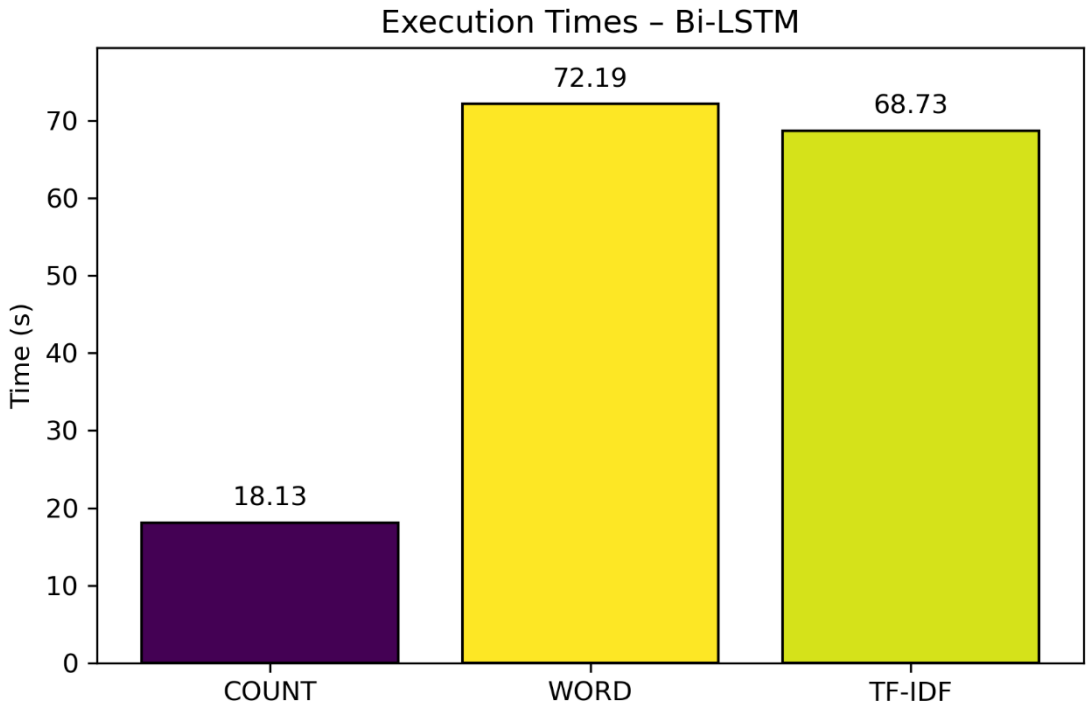


Figure 4.81 : Time execution of Bi-LSTM of Dataset2

3-Hybrid Algorithms:

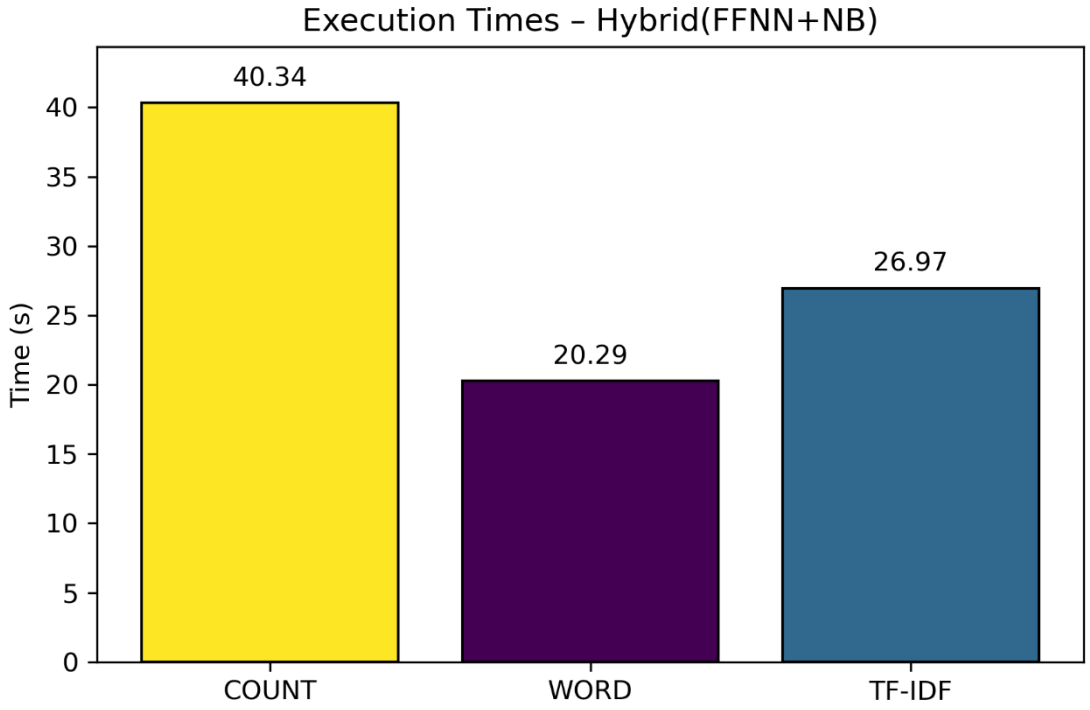


Figure 4.82 : Time execution of Hybrid FFNN+NB Dataset2

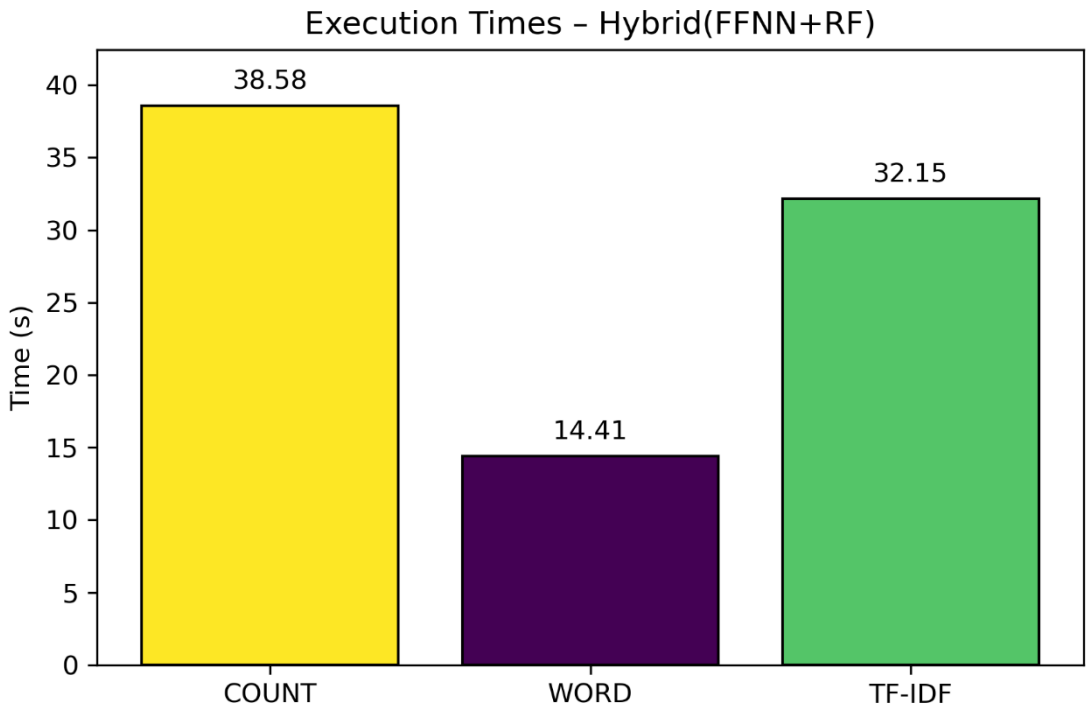


Figure 4.83 : Time execution of Hybrid FFNN+RF Dataset2

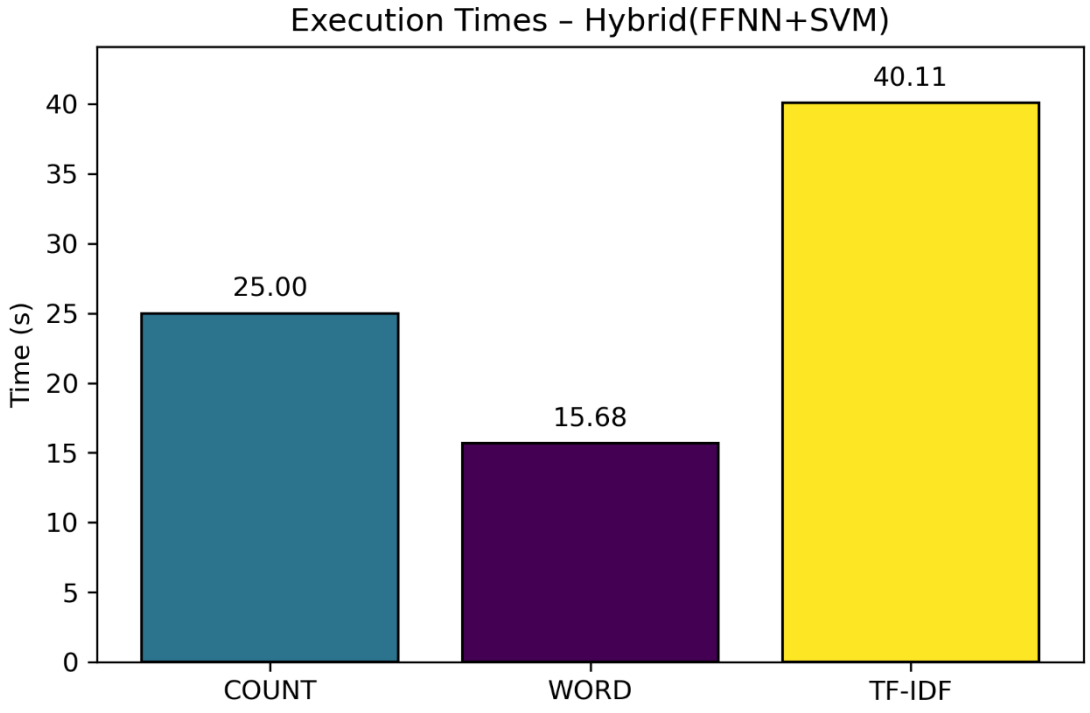


Figure 4.84 : Time execution of Hybrid FFNN+SVM Dataset2

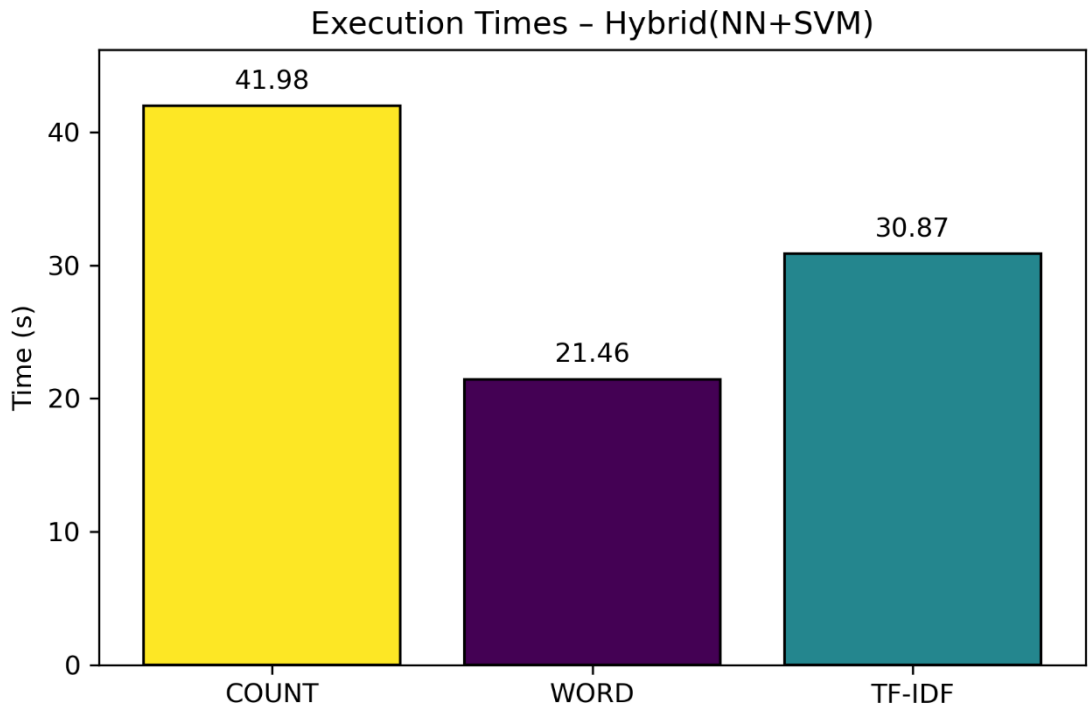


Figure 4.85 : Time execution of Hybrid NN+SVM Dataset2

4.2.3.3 Accuracy and Execution Time Comparison of Algorithms

Here, I compare the use of Word Vectorizer, Count Vectorizer, and TF-IDF and analyze the trade-off between accuracy and execution time for each calculation, taking into account both datasets. This comparison aids in offering a thorough comprehension of each calculation's precision and effectiveness. To provide a clear and intuitive ranking from most to least feasible, visual representations are used to indicate the trade-off—or necessity—between accuracy and execution time for all the calculations attempted. Determining which calculations offer the best trade-off becomes easier when these trade-offs are presented in a single chart for each dataset, especially when dealing with high-dimensional highlights produced by vector techniques. This study also demonstrates the evolving computational complexity and capabilities of both traditional calculations and deep learning, as well as hybrid algorithms, providing insights into their applicability to various types of text data.

Dataset1:

1-The following figure shows the accuracy and execution time of the Count Vectorization:

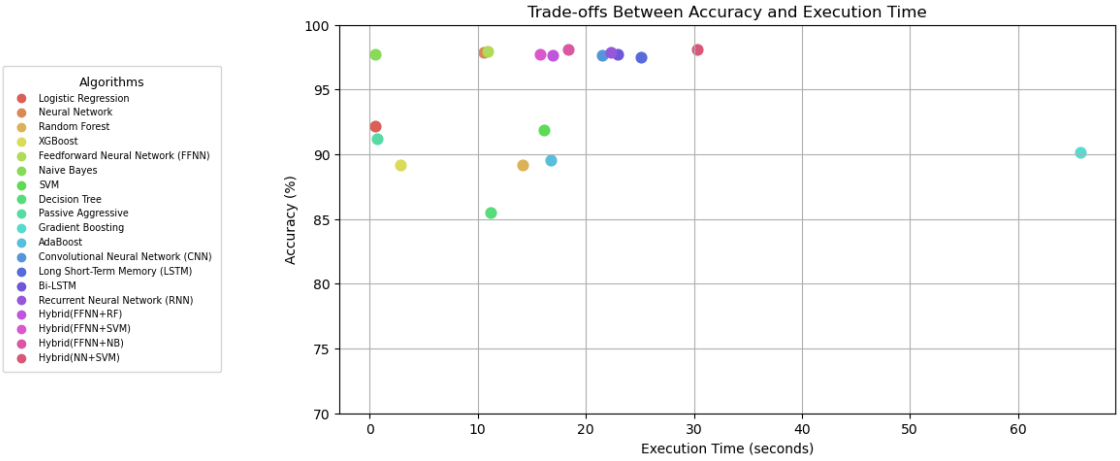


Figure 4.86 :Accuracy and Time Execution to used Count Vectorization of Dataset1

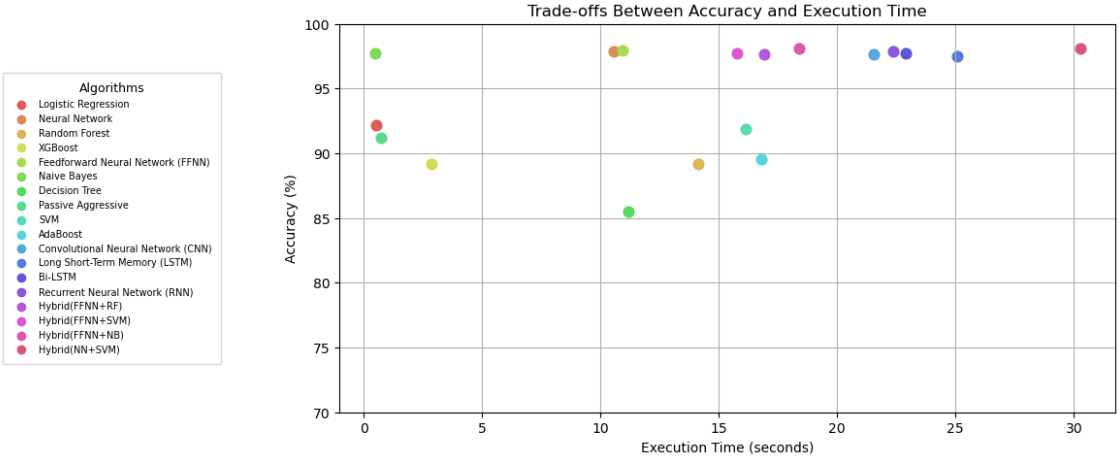


Figure 4.87: Accuracy and Time Execution to used Count Vectorization of Dataset1 without GB

The best algorithm is Naive Bayes and the worst is Decision Tree (low accuracy) and Gradient Boosting (time execution is very high).

2-The following figure shows the accuracy and execution time of the Word Vectorization**:**

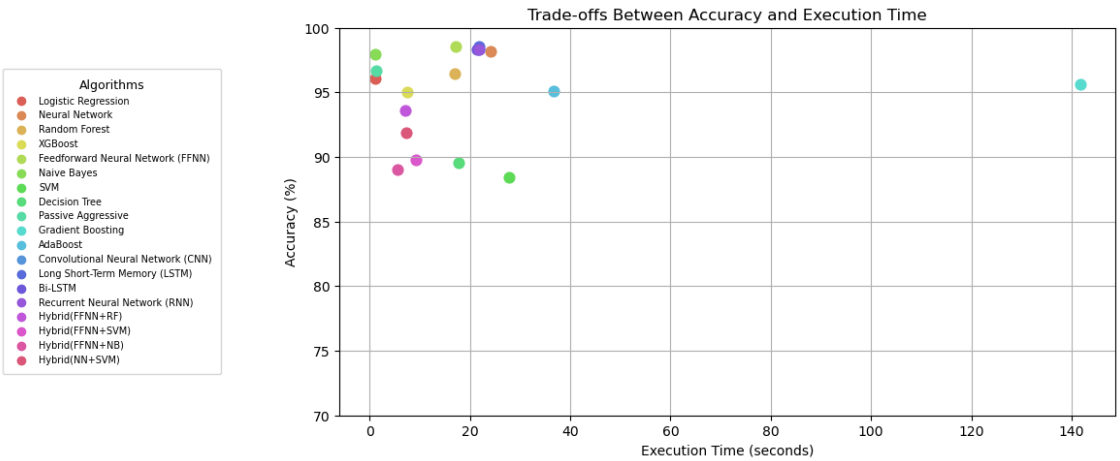


Figure 4.88 : Accuracy and Time Execution to used Word Vectorization of Dataset1

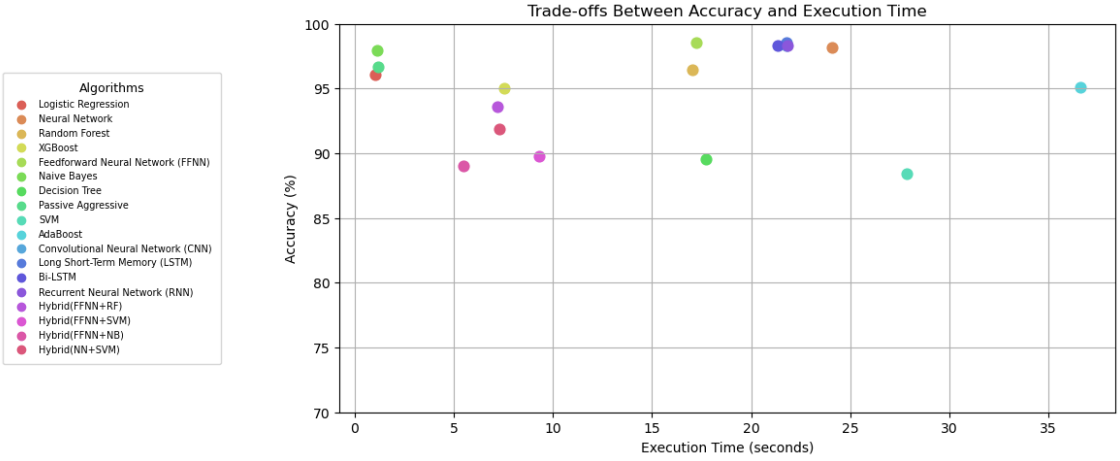


Figure 4.89 : Accuracy and Time Execution to used Word Vectorization of Dataset1 without GB

The best algorithm is Naive Bayes and the worst is Decision Tree and SVM (low accuracy) and Gradient Boosting (time execution is very high).

3- The following figure shows the accuracy and execution time of the TF-IDF:

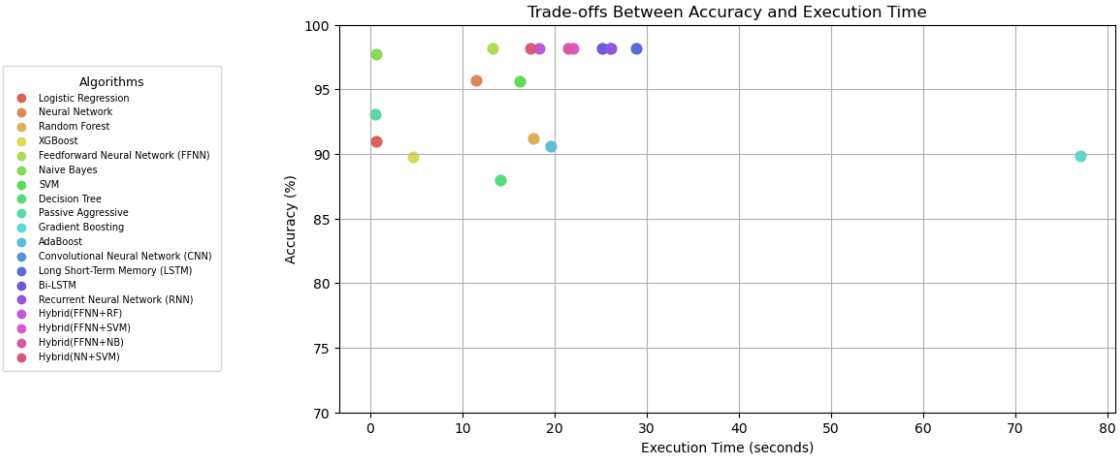


Figure 4.90: Accuracy and Time Execution to used TF-IDF of Dataset1

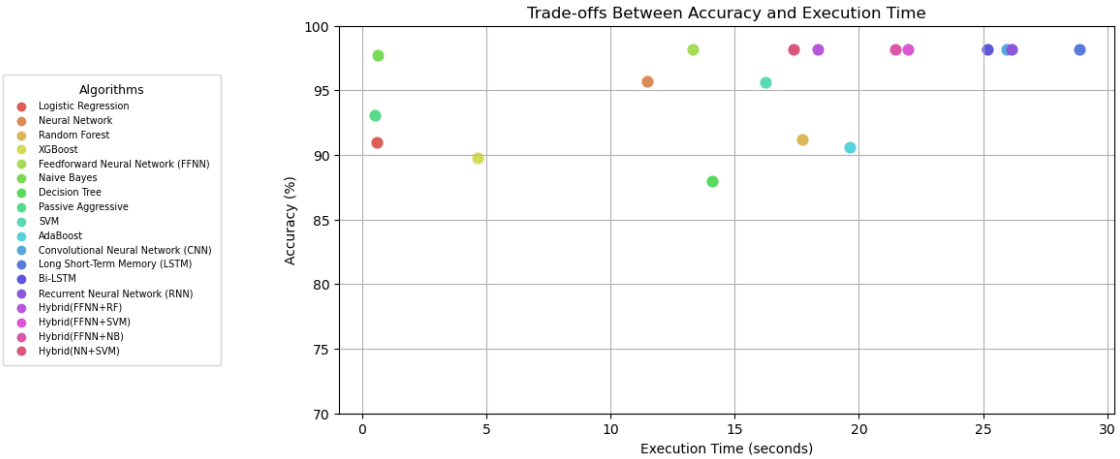


Figure 4.91: Accuracy and Time Execution to used TF-IDF of Dataset1 without GB

The best algorithm is Naive Bayes and the worst is Decision Tree and SVM (low accuracy) and Gradient Boosting (time execution is very high).

Dataset2:

1-The following figure shows the accuracy and execution time of the Count Vectorization:

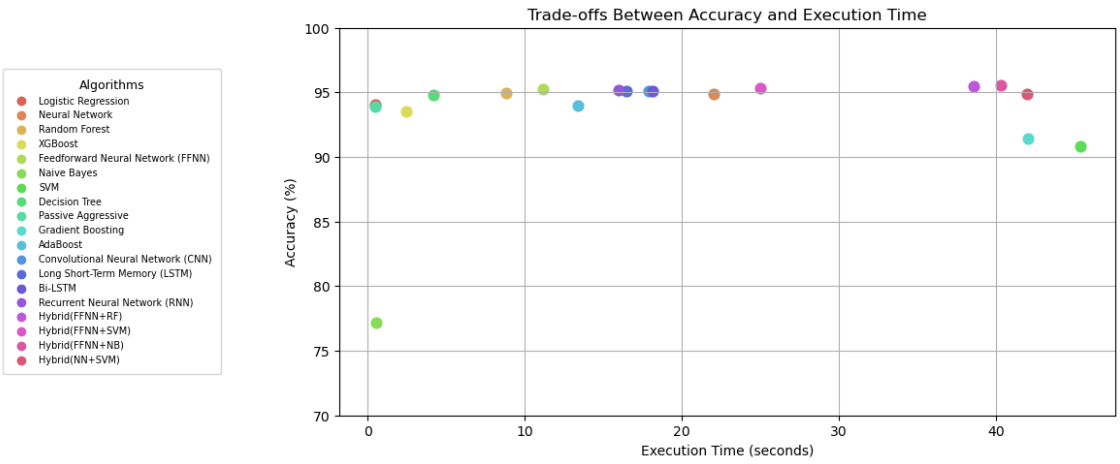


Figure 4.92 : Accuracy and Time Execution to used Count Vectorization of Dataset2

The best algorithm is Logistic Regression and FFNN, NN and the worst is Naive Bayes (low accuracy) and Gradient Boosting and SVM (time execution is very high).

2-The following figure shows the accuracy and execution time of the Word Vectorization:

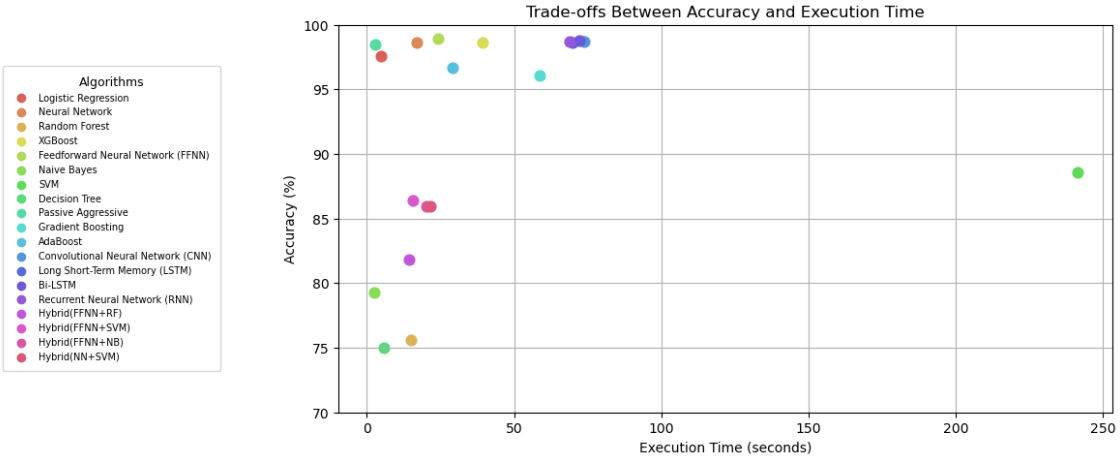


Figure 4.93 :Accuracy and Time Execution to used word Vectorization of Dataset2

Since the execution time of SVM algorithm is long, it does not clearly show the differences between other algorithms, so it was excluded and this Figure is shown.

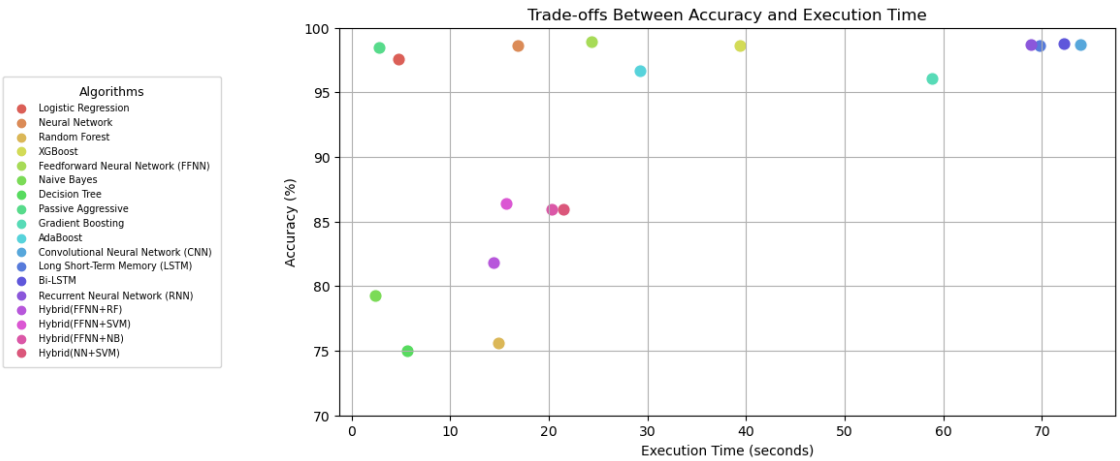


Figure 4.94 : Accuracy and Time Execution to used Word Vectorization of Dataset2 Without SVM

The best algorithm is Logistic Regression, Passive aggressive and NN and the worst is Naive Bayes, Random Forest and Decision Tree (low accuracy) and Gradient Boosting and SVM (time execution is very high).

3- The following figure shows the accuracy and execution time of the TF-IDF:

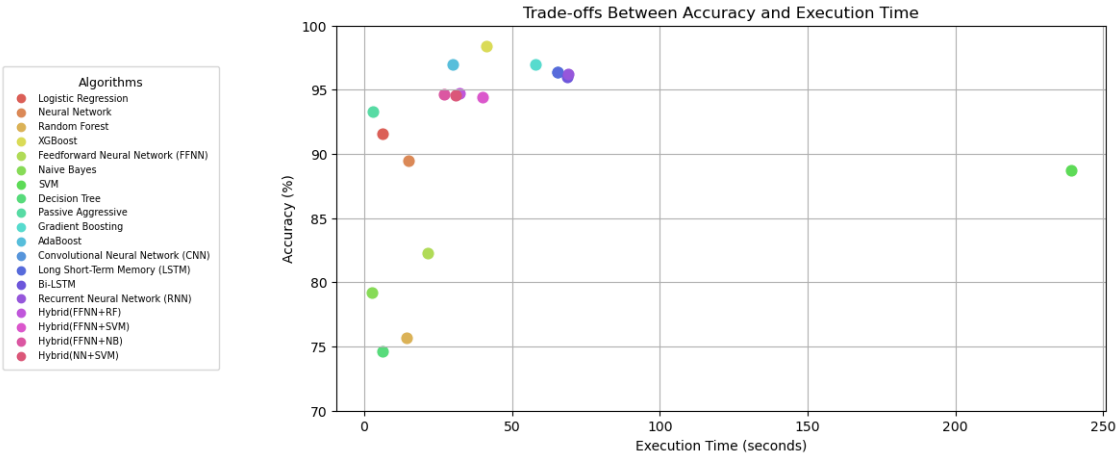


Figure 4.95: Accuracy and Time Execution to used TF-IDF of Dataset2

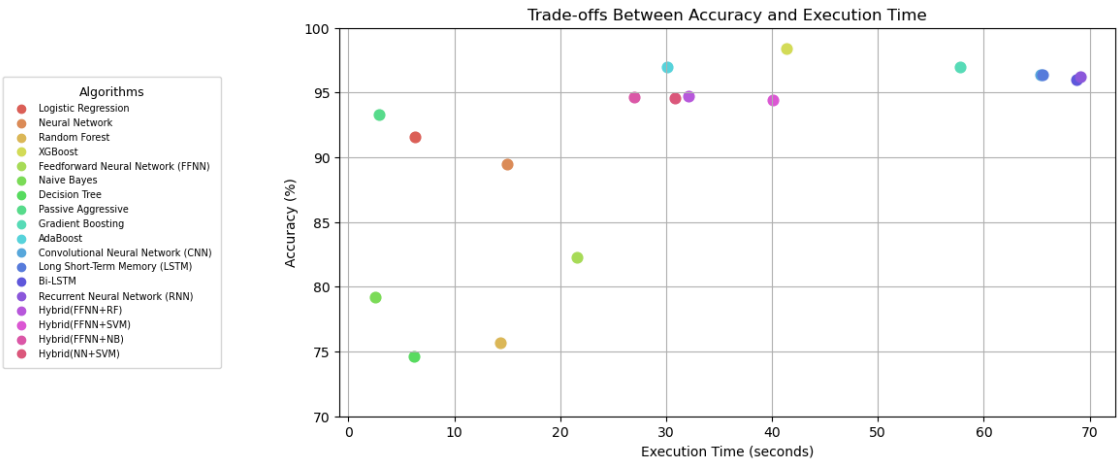


Figure 4.96: Accuracy and Time Execution to used TF-IDF of Dataset2 without SVM

The best algorithm is XGBoost, FFNN and the worst is Naive Bayes, Random Forest and Decision Tree (low accuracy) and Gradient Boosting and SVM (time execution is very high).

4.2.4 Analysis of Results

4.2.4.1 Accuracy, Precision, Recall, and Execution Time Analysis as Dataset 1

1- Count Vectorizer Results:

1. Top-performing models:

    1) With an execution time of 18.4 seconds and an F1-score of 0.969, Hybrid FFNN + Naive Bayes achieved an accuracy of 98.1%.
    2) An F1-score of 0.969 and 98.1% accuracy were similarly attained with hybrid NN + SVM, albeit with a longer execution time of 30.3 seconds.
    3) These findings imply that while employing straightforward frequency-based representations like Count Vectorizer, performance can be considerably improved by integrating deep learning models like neural networks with conventional classifiers (such Naive Bayes or SVM).

2. Lowest-performing model:

    1) Decision Tree recorded the most reduced accuracy at 85.5% with an F1-score of 0.805, demonstrating its restrictions in taking care of high-dimensional meager information produced by Count Vectorizer.

3. Execution Time Analysis:

    1) Models such as Naive Bayes and Logistic Regression had exceptionally brief execution times (0.46s and 0.52s, separately), making them appropriate for time-sensitive applications.
    2) In differentiate, SVM required 16.1 seconds, likely due to its computational complexity in high-dimensional space, which is ordinary with scanty vectors.

2-Word Vectorizer Results:

1. Top-performing model: With a 98.9% accuracy rate, an F1-score of 0.989, and an execution time of 24.3 seconds, FFNN performed best. This illustrates how deep learning models can make use of Word2Vec's semantic word embeddings to represent and classify text in a more meaningful way.
2. Lowest-performing models: Random Forest and Decision Tree showed poor performance (75.6% and 75.0% accuracy, respectively), likely due to their inability to effectively model the continuous and dense feature vectors produced by Word2Vec.
3. Execution Time Analysis: FFNN maintained a reasonable execution time while achieving superior performance, making it an effective choice for applications that prioritize both accuracy and efficiency.

3-TF-IDF (Term Frequency-Inverse Document Frequency):

1. Top-performing model:

1) XGBoost achieved 98.4% accuracy and an F1-score of 0.984, with an execution time of 41.35 seconds.
2) This shows that XGBoost is particularly effective in capturing complex interactions in high-dimensional sparse data generated by TF-IDF. It is well-suited for structured learning with tabular features.

2. Lowest-performing model: With an accuracy of just 79.2%, Naive Bayes performed poorly, most likely as a result of its reliance on feature independence, which is not reliable in TF-IDF representations where feature correlations are important.
3. Execution Time Analysis: In cases where high precision is crucial, XGBoost's performance justified the trade-off, even though it took longer.

-Summary of the previous part is as follows:

1. Count Vectorizer offers solid performance when paired with hybrid models, despite being a basic frequency-based technique.
   Word2Vec enhances performance significantly with deep learning models due to its semantic richness and contextual representation.
2. TF-IDF performs best with complex models like XGBoost, which are able to exploit intricate feature interactions.
3. Naive Bayes, while computationally efficient, generally underperforms on complex representations due to its naive assumptions.
4. SVM, although accurate, suffers from longer execution times on high-dimensional data, which may affect scalability.

4.2.4.2 Accuracy, Precision, Recall, and Execution Time Analysis as Dataset 2

We used a number of assessment criteria, including accuracy, precision, recall, F1-score, and execution time, to assess how well the applied machine learning and deep learning models and hybrid models performed in SQL injection detection. Together, these measures offer a more thorough understanding of each model's advantages and disadvantages under various text vectorization strategies. A thorough analysis of the

outcomes using the three widely used vectorization methods—Count Vectorizer, TF-IDF, and Word2Vec (Word Vectorizer)—is provided below.

1-Count vectorization:

The Count Vectorizer represents textual data as a matrix of token counts, which is simple yet often effective in traditional machine learning pipelines.

1. Top Performers: The best performance was obtained by FFNN + NB (Feedforward Neural Network with Naive Bayes), which had a very high F1-score of 0.955 and an accuracy of 95.55%.

   With 95.5% accuracy and similarly high F1-scores, FFNN + RF (Feedforward Neural Network with Random Forest) came in second.

   These findings imply that integrating the probabilistic or ensemble nature of NB and RF with the deep feature extraction powers of FFNN greatly enhances hybrid models. Even with basic frequency data, neural networks can grasp intricate correlations between tokens.
2. Underperformers: The Naive Bayes (NB) classifier alone recorded a eminently lower exactness of 77.15%, in spite of the fact that its review was exceptionally tall at 98.93%.

   This recommends that NB was able to accurately recognize most of the positive cases (delicate to genuine positives), but at the taken a toll of a tall false-positive rate (i.e., moo exactness).

   This awkwardness may be a known issue with NB, which assumes feature independence — a tricky suspicion for natural language data were word connections matter altogether.
3. Execution Time: Logistic Regression and Passive Aggressive Classifier completed predictions very quickly, with execution times around 0.44–0.45 seconds, making them efficient for real-time detection use cases.

   On the other hand, Support Vector Machines (SVM) took significantly longer (~45.37 seconds), possibly due to the high-dimensional sparse data generated by Count Vectorizer.

   The high computational cost of SVMs, especially with large datasets and many features, can make them impractical for time-critical environments.

2-Word Vectorization:

Word2Vec provides dense, semantic vector representations for each word by training on context windows. This method captures both syntactic and semantic relationships between words:

1- Top Performers: The FFNN (Feedforward Neural Network) model, when applied to Word2Vec representations, achieved the highest performance across all configurations, with an outstanding accuracy of 98.9%, F1-score of 0.989, and execution time of 24.29 seconds.
   This demonstrates the strength of deep learning in capturing intricate relationships in semantic space, which is not possible with frequency-based approaches like Count Vectorizer.
2- Underperformers: Traditional tree-based models such as Random Forest and Decision Tree performed poorly, with accuracies of 75.6% and 75.0%, respectively.
   These models typically struggle with dense, continuous vector inputs that do not follow the hierarchical, discrete structure they are optimized for.
   Additionally, Word2Vec introduces feature interdependence and context sensitivity — factors that decision trees cannot leverage effectively.
3- Execution Time: The FFNN model demonstrated a good balance between accuracy and computation, with execution times under 25 seconds — acceptable for most real-world scenarios.

3- TF-IDF (Term Frequency-Inverse Document Frequency):

By penalizing frequently recurring words that might not include important information, TF-IDF improves on basic count-based vectorization and provides a more informative and balanced representation.

1- Top Performers: The top-performing model was XGBoost (Extreme Gradient Boosting), which had a moderate execution time of 41.35 seconds, a 0.984 F1-score, and an accuracy of 98.4%.
   This illustrates how well XGBoost can handle intricate feature interactions, generate precise predictions, and efficiently handle sparse data.

Because of its regularization and tree-based structure, it works particularly well with structured/tabular data, even when it is produced from unstructured text.

2- Underperformers: Naive Bayes once more slacked behind with an accuracy of 79.2%, likely due to the same autonomy presumptions being abused in TF-IDF frameworks. Since TF-IDF emphasizes rarer terms, this shifts the conveyance of highlights, which may advance corrupt NB's execution.
3- Execution Time: Whereas XGBoost was more computationally seriously than less difficult models like Logistic Regression or NB, its predominant exactness legitimizes its taken a toll in numerous utilize cases, particularly where prescient control may be a need over speed.

Summary of the previous part is as follows:

1- When used to semantic embeddings such as Word2Vec, Deep Learning Models (particularly FFNN) perform exceptionally well, exhibiting excellent accuracy and comparatively low computing costs.
2- With frequency-based inputs, hybrid models that combine neural networks and conventional classifiers (such as FFNN + NB or FFNN + RF) capitalize on the advantages of both methodologies and yield better results.
3- Because of its strong handling of complex, sparse data, XGBoost stands out among conventional models, particularly when TF-IDF is used.
4- Even while Naive Bayes is quick and easy to use, it performs poorly in terms of accuracy and F1-score, particularly where word dependencies are important.
5- The needs of the application should determine which model is best; if real-time prediction is crucial, quick models like logistic regression can be better. The trade-offs for high-accuracy applications like threat detection are better with FFNN or XGBoost.

4.2.4.3 Specific Observations for Algorithms:

1. Logistic Regression: This calculation has dependably outlined tall accuracy for as of presently coordinate models when utilizing Word Vectorizer Its

straightforwardness and capability ensure sensible execution times making it a strong choice for errands with moo computational budgets.

2. Neural Network: The neural network fulfilled slight accuracy improvements with the Word Vectorizer showing its capability to handle complex plans inside the data. In spite of these picks, the computational get remained sensible compared to customary machine learning techniques.
3. Random Forest: Exactness progressed for Dataset 1 but diminished for Dataset 2 when utilizing the Word Vectorizer. This proposes that Random Forest's execution is affected by dataset-specific qualities such as class distribution and feature variability.
4. XGBoost: The calculation accomplished great exactness with the Word Vectorizer but at a critical computational cost. The drawn-out execution times can be credited to the serious gradient-boosting preparation, especially when taking care of high-dimensional information. Utilizing parallel preparation or leveraging equipment to increase speed e.g. GPUs seem to altogether diminish this burden.

4.2.5 Exclusion of Certain Algorithms

Due to the long training and testing periods, Gradient Boosting and SVM were not included in the final results. The computational costs of these approaches were considered inefficient in this research, given the already high accuracy rates of other models, for example the accuracy of the Passive Aggressive algorithm is higher than the two algorithms, in addition to the very short execution time (approximately one second).

4.2.6 Key Results

4.2.6.1 Accuracy

The deep learning-based models, specifically the Feedforward Neural Network (FFNN) and Neural Network (NN), reliably obtained the greatest accuracy rates across both datasets, especially when combined with Word2Vec and TF-IDF vectorization approaches. The robustness of the FFNN across several input formats was demonstrated in Dataset 2, where it achieved an accuracy of 98.9% with Word2Vec and 95.25% with

Count Vectorizer. On the other hand, hybrid models such as FFNN + NB and FFNN + RF demonstrated notable gains in accuracy using Count Vectorizer, attaining 95.5% and 95.5%, respectively.

Traditional machine learning models, counting Naive Bayes and Logistic Regression, too performed sensibly well, especially with Word2Vec, with Logistic Regression coming to 97.7curacy and a solid F1-score of 0.977. On the other hand, tree-based algorithms such as Random Forest and Decision Tree shown lower accuracy in a few setups, particularly when combined with TF-IDF, showing affectability to vectorization methods.

In general, Word2Vec vectorization led to enhanced classification accuracy for most algorithms, with clear advantages for deep learning and hybrid approaches.

4.2.6.2 Precision, Recall, and F1-Score

The trends in accuracy were reflected in the F1-score, a crucial indicator for unbalanced data. Top-tier F1-scores above 0.95 were attained by deep learning models like FFNN and Bi-LSTM using all vectorization techniques. For example, the FFNN obtained an F1-score of 0.9526 with Count Vectorizer and 0.989 with Word2Vec.Hybrid models also maintained strong F1-scores when combined with Count Vectorizer, such as FFNN + NB (0.955) and FFNN + RF (0.954), showing their capability to balance precision and recall effectively.

In contrast, while models like Decision Tree showed relatively high recall, their low precision values brought down the F1-score, particularly with TF-IDF (F1-score ≈ 0.754). These findings emphasize that high recall alone is insufficient without accompanying precision in security-sensitive tasks like SQL injection detection.

4.2.6.3 Execution Time

The model and vectorization technique had a substantial impact on execution time. In general, Word2Vec resulted in lengthier training periods, especially for intricate

models like FFNN (24.3 seconds) and XGBoost (42.1 seconds). In many situations, the accuracy and F1-scores outweigh these delays.
With execution speeds as low as 0.3 to 0.45 seconds, Naive Bayes and Logistic Regression, on the other hand, continued to be the most computationally efficient algorithms among all vectorizers, making them appropriate for applications where speed is crucial.

Count Vectorizer had the best trade-off between execution time and accuracy, especially for hybrid models. TF-IDF, while producing competitive results with deep models like XGBoost (98.4% accuracy, 0.984 F1-score), generally required more processing time than Count Vectorizer and less than Word2Vec.

4.2.6.4 Overall Comparison

The combination of FFNN with Word2Vec given the most excellent generally execution over datasets, yielding the most elevated accuracy and F1-score, in spite of the fact that at the fetched of expanded computation time. Hybrid models like FFNN + NB and FFNN + RF advertised solid options with quicker preparing times, particularly when utilized with Count Vectorizer. In differentiate, less difficult models such as Naive Bayes and Logistic Regression conveyed palatable comes about in altogether less time, making them practical choices for real-time frameworks with compelled assets.

4.2.6.5 Vectorization Technique Impact

The choice of vectorization technique had a big impact on model performance. Word2Vec increased the accuracy of both deep and conventional models, but it also caused noticeable training time delays. Count Vectorizer improved the performance of hybrid models in particular while producing consistent and balanced results for all model types. For XGBoost and Bi-LSTM, TF-IDF performed better, but it was less successful with simpler tree-based models.

4.2.6.6 Conclusion

This thorough research demonstrates how the algorithm and the text representation method have a significant impact on the model's performance. Particularly with Word2Vec and Count Vectorizer, deep learning and hybrid models showed better accuracy and F1-scores. In real-world applications, their higher computing cost must be taken into account. Depending on the particular needs of the SQL injection detection scenario, choosing the right vectorization approach and model is therefore crucial to striking an effective balance between computational efficiency and predictive performance.

5. Chapter Five: Conclusions and Future Directions

A thorough overview of the results pertaining to SQL injection detection utilizing conventional, deep learning, and hybrid machine learning algorithms is provided in this chapter. Using two different datasets, the study examined different models according to their accuracy, precision, recall, F1-score, and execution time. Furthermore, the chapter addresses the trade-offs between performance and computational complexity, while offering practical directions for future work.

Throughout the study, several algorithms were implemented and tested, including deep learning models such as Feedforward Neural Network (FFNN), Neural Network (NN), LSTM, Bi-LSTM, and traditional models such as Naïve Bayes, Random Forest, XGBoost, Support Vector Machine (SVM), and Logistic Regression. Additionally, four hybrid models were designed and analyzed: FFNN+RF, FFNN+SVM, FFNN+NB, and NN+SVM.

The major conclusions are summarized as follows:

5.1 Key Findings

The evaluation of both traditional and deep learning algorithms revealed several critical insights:

1-The Accuracy of Deep Learning Algorithms Was Superior

- Across both datasets, deep learning models continuously beat conventional techniques in terms of accuracy.
- On Dataset 1 and Dataset 2, FFNN obtained 97% and 95% accuracy, respectively. Similarly, the accuracy of NN was 98% and 94.5%, respectively.
- These findings show how well deep architectures can identify SQL injection patterns.

2-Inconsistent Performance of Naive Bayes

- Naïve Bayes delivered excellent accuracy (97.5%) on Dataset1 but dropped significantly (77%) on Dataset2.

- This variety recommends that Naïve Bayes is touchy to course awkwardness and the dissemination of highlights, making it less dependable in certain settings.

3-Vital Highlights Reliably Recognized

- Particular terms such as “select,” “admin,” and “union” were reliably recognized as basic pointers of SQL infusion by multiple algorithms.
- This consistency underscores the importance of effective feature selection and vectorization.

4-Performance Is Affected by the Vectorization Technique

- Generally speaking, Word Vectorizer (such as Word2Vec) increased model accuracy; but, because of high-dimensional data, it resulted in lengthier training and testing periods.
- Although it was computationally efficient, Count Vectorizer's accuracy was marginally inferior.
- By offering a performance balance between speed and accuracy, TF-IDF provided a compromise.

5-Execution Time Trade-Offs

- Traditional calculations like Naïve Bayes and Logistic Regression executed in less than one second, making them reasonable for real-time applications.
- In differentiate, deep learning models such as LSTM and BiLSTM accomplished high accuracy but required essentially longer execution time.
- Hybrid models such as FFNN+SVM and NN+SVM balanced performance and efficiency, offering promising results for practical deployment.

5.2 Alignment with Research Objectives

The findings directly address the research questions and objectives as follows:

1) Effectiveness of Algorithms

   According to the study, FFNN and NN were the most effective algorithms for identifying SQL injection. By utilizing their ability to recognize intricate patterns in text data, these models help meet Objective 1 by producing high accuracy, precision, recall, and F1-scores. This is especially noticeable in Dataset1, where NN achieved 98% accuracy and FFNN obtained 97%.

2) Dataset Diversity and Generalizability

Testing the models on two free datasets illustrated the versatility of deep learning and hybrid models. Whereas traditional models such as Naïve Bayes appeared tall execution on Dataset1, they dropped essentially on Dataset2 (from 97.5% to 77%), uncovering restricted generalizability. In contrast, hybrid models like FFNN+SVM and NN+SVM maintained stable performance, confirming Objective 2: ensuring robustness across varied datasets.

3) Recommendations for Improvement

The observed limitations of some traditional models inspired the development of hybrid models, such as FFNN+NB and NN+SVM, which combine the strengths of different architectures to improve reliability and accuracy. These approaches align with Objective 3 by providing pathways to overcome the precision and recall limitations of individual models.

Additionally, the ponder emphasized utilizing numerous assessment measurements — not fair accuracy, but too precision, recall, F1-score, and execution time — to supply a more comprehensive appraisal of demonstrate execution.

5.3 Trade-Off and Limitation

The study identified several trade-offs between accuracy, robustness, and computational efficiency:

1- Deep Learning Models:

1) Advantages: Accomplished high accuracy and strong precision/recall values. NN and FFNN models appeared exceptional F1-scores and steadiness over datasets.
2) Limitations: High computational fetched, longer training/testing time, and request for high-performance equipment (e.g., GPUs).

2- Traditional Algorithms:

1) Advantages: Faster execution times (often under 1 second), low resource requirements, and ease of implementation.

2) Limitations: Lower accuracy and F1-scores, particularly under high-dimensional and unbalanced data distributions.

3- Vectorization Techniques:

1. Count Vectorizer is quick and efficient, although it does a moderate job of collecting contextual relationships.
2. Because of its increased dimensionality, Word Vectorizer (Word2Vec) required more processing time, but it improved accuracy and precision.
3. TF-IDF: Offered a fair compromise between execution time, recall, and accuracy. With less computational overhead than Word2Vec, it provided better context representation than Count Vectorizer.

4- Hybrid Models:

1. Appeared adjusted execution, leveraging the learning control of deep networks and the decision strength of traditional classifiers. For case, FFNN+SVM and NN+SVM given steady F1-scores and decreased execution time compared to standalone deep models. These Hybrid Models are promising for real-world arrangement where precision and speed are both basic.

5.4 Recommendations for Future Work

1- Enhancement of Hybrid Models

Assist investigate ought to investigate optimized combinations such as FFNN+RF, FFNN+NB, and NN+SVM. These models appeared eminent enhancements in exactness, review, and F1-score whereas adjusting execution time.

2- Applications in Real Time

Create lightweight deep learning or hybrid models that are appropriate for real-time SQL injection detection, where prompt threat mitigation depends on low latency and high recall.

3- Increasing the Diversity of Datasets

   To improve generalizability and resilience, include bigger and more intricate datasets, such as actual online traffic and sophisticated SQL injection variants (such polymorphic or blind SQLi).

4- Using Cutting-Edge Methods

   Use attention techniques, transfer learning, and adversarial training to strengthen the model's resistance to new SQLi attack patterns.

5- Increased Efficiency in Computation

   When resources are few, use TF-IDF as a sensible and well-balanced vectorization technique. Examine parallel processing and hardware acceleration (such as GPUs and TPUs) to reduce training time, especially for deep and hybrid models.

6- Evaluation Using Multiple Measures

   In addition to accuracy, future models should be regularly assessed using precision, recall, F1-score, and execution time to give a more complete picture of the model's dependability and environment compatibility.

5.5 Final Thoughts

Deep learning algorithms consistently showed the potential to detect SQL injection attacks with remarkable precision, especially when evaluated across multiple metrics, including accuracy, precision, recall, and F1-score, in contrast to traditional algorithms that frequently performed exceptionally well on one dataset but struggled on another. Although efficient in terms of execution time, traditional models like Naive Bayes and Logistic Regression displayed notable dataset-specific performance fluctuations, suggesting their poor capacity to generalize across various attack types and data representations.

In terms of execution time, deep learning models like FFNN and NN exhibited higher computational costs compared to traditional models. However, they provided outstanding precision and robust performance, especially on Dataset1. While these models require significant computational resources and longer processing times, their superior recall, ability to capture complex patterns, and strong F1-scores make them ideal candidates for applications where accuracy is the top priority, despite the need for enhanced infrastructure.

In addition, the utilize of hybrid algorithms such as FFNN+SVM, FFNN+NB, and NN+SVM gives an ideal adjust between accuracy, precision, and execution time. These hybrid models combine the qualities of both deep learning and traditional algorithms, advertising a critical advantage in scenarios where both tall execution and asset proficiency are required.

Looking ahead, deep learning algorithms offer unmatched flexibility and potential, clearing the way for more vigorous and secure web applications competent of guarding against a more extensive run of cyber threats. Their capacity to handle complex designs in high-dimensional information permits them to exceed expectations in distinguishing SQL injection assaults over assorted assault vectors. Hybrid models, combining the leading of both universes, might advance improve execution, advertising quicker location times and higher review rates whereas keeping up strong precision.

Future research should concentrate on improving these models for real-time SQL injection detection applications, with a particular emphasis on low-latency and scalable operations. Furthermore, their full potential would be realized if they were used to address a range of complex cybersecurity issues (such as polymorphic SQL injections and Zero-Day attacks). The performance of hybrid models suggests that they should be actively investigated and enhanced to maximize computing efficiency and further enhance detection accuracy.

References

Abdullah, H. S., & Abdulazeez, A. M. (2024). Detection of SQL Injection Attacks Based on Supervised Machine Learning Algorithms: A Review. *International Journal of Informatics, Information System and Computer Engineering (INJIISCOM)*, *5*(2), 152-165.

Abebe, A., Belay, Y., Belay, A., & Gebeyehu, S. (2024). SQL Injection Attacks Detection: A Performance Comparison on Multiple Classification Models. *Ethiopian International Journal of Engineering and Technology*, *2*(1), 22-38.

AL-Maliki, M. H. A., & Jasim, M. N. (2022). Review of SQL injection attacks: Detection, to enhance the security of the website from client-side attacks. *International Journal of Nonlinear Analysis and Applications*, *13*(1), 3773-3782.

Alanda, A., Satria, D., Ardhana, M. I., Dahlan, A. A., & Mooduto, H. A. (2021). Web application penetration testing using SQL Injection attack. *JOIV: International Journal on Informatics Visualization*, *5*(3), 320-326.

Alarfaj, F. K., & Khan, N. A. (2023). Enhancing the performance of SQL injection attack detection through probabilistic neural networks. *Applied Sciences*, *13*(7), 4365.

Alenezi, M., Nadeem, M., & Asif, R. (2021). SQL injection attacks countermeasures assessments. *Indonesian Journal of Electrical Engineering and Computer Science*, *21*(2), 1121-1131.

Alghawazi, M., Alghazzawi, D., & Alarifi, S. (2022). Detection of sql injection attack using machine learning techniques: a systematic literature review. *Journal of Cybersecurity and Privacy*, *2*(4), 764-777.

Alghawazi, M., Alghazzawi, D., & Alarifi, S. (2023). Deep learning architecture for detecting SQL injection attacks based on RNN autoencoder model. *Mathematics*, *11*(15), 3286.

AlShammari, A. F. Implementation of Keyword Extraction using Term Frequency-Inverse Document Frequency (TF-IDF) in Python. *International Journal of Computer Applications*, *975*, 8887.

Ángeles Rojas, J. A., Calderon Vilca, H. D., Tumi Figueroa, E. N., Cuadros Ramos, K. J., Matos Manguinuri, S. S., & Calderón Vilca, E. F. (2021). Hybrid model of convolutional neural network and support vector machine to classify basal cell carcinoma. *Computación y Sistemas*, *25*(1), 83-95.

Arnap, A. (2024). Enhancing SQL Injection Attack Detection Using Naïve Bayes and SMOTE Method on Imbalanced Datasets. *Journal of Artificial Intelligence and Engineering Applications (JAIEA)*, *4*(1), 74-81.

Arsyah, U. I., Pratiwi, M., & Muhammad, A. (2024). Twitter Sentiment Analysis of Public Space Opinions using SVM and TF-IDF Methods. *The Indonesian Journal of Computer Science*, *13*(1).

Bakush, M. M. (2024). https://ieeexplore. ieee. org/abstract/document/9464453.

Bommasani, R., Davis, K., & Cardie, C. (2020). Interpreting pretrained contextualized representations via reductions to static embeddings. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics,

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., & Askell, A. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877-1901.

Brownlee, J. (2022). *Machine learning mastery*. Machine Learning Mastery.

Chen, C., Zhang, P., Zhang, H., Dai, J., Yi, Y., Zhang, H., & Zhang, Y. (2020). Deep learning on computational-resource-limited platforms: A survey. *Mobile Information Systems*, *2020*(1), 8454327.

Chen, D., Yan, Q., Wu, C., & Zhao, J. (2021). Sql injection attack detection and prevention techniques using deep learning. Journal of Physics: Conference Series,

Chollet, F. (2021). *Deep learning with Python*. Simon and Schuster.

Crespo-Martínez, I. S., Campazas-Vega, A., Guerrero-Higueras, Á. M., Riego-DelCastillo, V., Álvarez-Aparicio, C., & Fernández-Llamas, C. (2023). SQL injection attack detection in network flow data. *Computers & Security*, *127*, 103093.

Demilie, W. B., & Deriba, F. G. (2022). Detection and prevention of SQLI attacks and developing compressive framework using machine learning and hybrid techniques. *Journal of Big Data*, *9*(1), 124.

Deriba, F. G., Salau, A. O., Mohammed, S. H., Kassa, T. M., & Demilie, W. B. (2022). Development of a compressive framework using machine learning approaches for SQL injection attacks. *Przeglad Elektrotechniczny*, *98*(7), 181-187.

Djaballah, S., Saidi, L., Meftah, K., Hechifa, A., Bajaj, M., & Zaitsev, I. (2024). A hybrid LSTM random forest model with grey wolf optimization for enhanced detection of multiple bearing faults. *Scientific Reports*, *14*(1), 23997.

Farooq, U. (2020). Real time password strength analysis on a web application using multiple machine learning approaches. *International Journal of engineering research and technology*, *9*(12), 359-364.

Farooq, U. (2021). Ensemble machine learning approaches for detection of SQL injection attack. *Tehnički glasnik*, *15*(1), 112-120.

Flores Jr, C. P., & Monreal, R. N. (2024). Evaluation of Common Security Vulnerabilities of State Universities and Colleges Websites Based on OWASP. *Journal of Electrical Systems*, *20*(5s), 1396-1404.

Gallicchio, C., & Scardapane, S. (2020). Deep randomized neural networks. Recent Trends in Learning From Data: Tutorials from the INNS Big Data and Deep Learning Conference (INNSBDDL2019),

Gandhi, N., Patel, J., Sisodiya, R., Doshi, N., & Mishra, S. (2021). A CNN-BiLSTM based approach for detection of SQL injection attacks. 2021 International conference on computational intelligence and knowledge economy (ICCIKE),

Géron, A. (2019). Hands-on machine learning with scikit-learn, keras, and tensorflow: concepts. *Aurélien Géron-Google Kitaplar*, *yy https://books. google. com. tr/books*.

Ghosh, A., Diyasi, S., & Chatterjee, S. (2024). Enhancing SQL Injection Prevention: Advanced Machine Learning and LSTM-Based Techniques. *International Journal on Computational Modelling Applications*, *1*(1), 20-31.

Ghozali, I., Asy'ari, M. F., Triarjo, S., Ramadhani, H. M., Studiawan, H., & Shiddiqi, A. M. (2022). A novel SQL injection detection using Bi-LSTM and TF-IDF. 2022 7th International Conference on Information and Network Technologies (ICINT),

Goodfellow, I. (2016). Deep learning. In: MIT press.

Grootendorst, M. (2022). BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv preprint arXiv:2203.05794*.

Halfond, W. G., Viegas, J., & Orso, A. (2006). A Classification of SQL Injection Attacks and Countermeasures. ISSSE,

Hasan, M., Balbahaith, Z., & Tarique, M. (2019). Detection of SQL injection attacks: a machine learning approach. 2019 International Conference on Electrical and Computing Technologies and Applications (ICECTA),

Hassan, M. M., Ahmad, R. B., & Ghosh, T. (2021). SQL injection vulnerability detection using deep learning: a feature-based approach. *Indonesian Journal of Electrical Engineering and Informatics (IJEEI)*, *9*(3), 702-718.

Ibarra-Fiallos, S., Higuera, J. B., Intriago-Pazmiño, M., Higuera, J. R. B., Montalvo, J. A. S., & Cubo, J. (2021). Effective filter for common injection attacks in online web applications. *Ieee access*, *9*, 10378-10391.

Jemal, I., Cheikhrouhou, O., Hamam, H., & Mahfoudhi, A. (2020). Sql injection attack detection and prevention techniques using machine learning. *International Journal of Applied Engineering Research*, *15*(6), 569-580.

Jiao, Q., & Zhang, S. (2021). A brief survey of word embedding and its recent development. 2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC),

Jothi, K., Pandey, N., Beriwal, P., & Amarajan, A. (2021). An efficient SQL injection detection system using deep learning. 2021 International conference on computational intelligence and knowledge economy (ICCIKE),

Kakisim, A. G. (2024). A deep learning approach based on multi-view consensus for SQL injection detection. *International Journal of Information Security*, *23*(2), 1541-1556.

Kandel, I., & Castelli, M. (2020). The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset. *ICT express*, *6*(4), 312-315.

Kenton, J. D. M.-W. C., & Toutanova, L. K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. Proceedings of naacL-HLT,

Kingma, D. P. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krishna, H., Oluoch, J., & Kim, J. A Hybrid Approach for Detecting SQL-Injection Using Machine Learning Techniques.

Krishnan, S. A., Sabu, A. N., Sajan, P. P., & Sreedeep, A. (2021). SQL injection detection using machine learning. *Revista Geintec-Gestao Inovacao E Tecnologias*, *11*(3), 300-310.

Le, T.-T.-H., Hwang, Y., Choi, C., Wardhani, R. W., Putranto, D. S. C., & Kim, H. (2024). Enhancing Structured Query Language Injection Detection with Trustworthy Ensemble Learning and Boosting Models Using Local Explanation Techniques. *Electronics*, *13*(22), 4350.

Li, M., Soltanolkotabi, M., & Oymak, S. (2020). Gradient descent with early stopping is provably robust to label noise for overparameterized neural networks. International conference on artificial intelligence and statistics,

Li, Q., Wang, F., Wang, J., & Li, W. (2019). LSTM-based SQL injection detection method for intelligent transportation system. *IEEE Transactions on Vehicular Technology*, *68*(5), 4182-4191.

Li, Y., & Zhang, B. (2019). Detection of SQL injection attacks based on improved TFIDF algorithm. Journal of Physics: Conference Series,

Liengaard, B. D., Sharma, P. N., Hult, G. T. M., Jensen, M. B., Sarstedt, M., Hair, J. F., & Ringle, C. M. (2021). Prediction: coveted, yet forsaken? Introducing a cross-validated predictive ability test in partial least squares path modeling. *Decision Sciences*, *52*(2), 362-392.

Lilhore, U. K., Sharma, Y. K., Shukla, B. K., Vadlamudi, M. N., Simaiya, S., Alroobaea, R., Alsafyani, M., & Baqasah, A. M. (2025). Hybrid convolutional neural network and bi-LSTM model with EfficientNet-B0 for high-accuracy breast cancer detection and classification. *Scientific Reports*, *15*(1), 12082.

Liu, C., Gu, Z., & Wang, J. (2021). A hybrid intrusion detection system based on scalable k-means+ random forest and deep learning. *Ieee access*, *9*, 75729-75740.

Liu, H.-I., Galindo, M., Xie, H., Wong, L.-K., Shuai, H.-H., Li, Y.-H., & Cheng, W.-H. (2024). Lightweight deep learning for resource-constrained environments: A survey. *ACM Computing Surveys*, *56*(10), 1-42.

Liu, W., Wang, Z., Liu, X., Zeng, N., Liu, Y., & Alsaadi, F. E. (2017). A survey of deep neural network architectures and their applications. *Neurocomputing*, *234*, 11-26.

Luo, A., Huang, W., & Fan, W. (2019). A CNN-based Approach to the Detection of SQL Injection Attacks. 2019 IEEE/ACIS 18th International Conference on Computer and Information Science (ICIS),

Mars, M. (2022). From word embeddings to pre-trained language models: A state-of-the-art walkthrough. *Applied Sciences*, *12*(17), 8805.

Marwah, M., Narayanan, A., Jou, S., Arlitt, M., & Pospelova, M. (2024). Is F1 Score Suboptimal for Cybersecurity Models? Introducing Cscore, a Cost-Aware Alternative for Model Assessment.

McCormick, C. (2016). Word2vec tutorial-the skip-gram model. *Apr-2016.[Online]. Available: http://mccormickml. com/2016/04/19/word2vec-tutorial-the-skip-gram-model*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Muslihi, M. T., & Alghazzawi, D. (2020). Detecting SQL injection on web application using deep learning techniques: a systematic literature review. 2020 Third International Conference on Vocational Education and Electrical Engineering (ICVEE),

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP),

Peralta-Garcia, E., Quevedo-Monsalbe, J., Tuesta-Monteza, V., & Arcila-Diaz, J. (2024). Detecting Structured Query Language Injections in Web Microservices Using Machine Learning. Informatics,

Phinzi, K., Abriha, D., & Szabó, S. (2021). Classification efficacy using k-fold cross-validation and bootstrapping resampling techniques on the example of mapping complex gully systems. *Remote Sensing*, *13*(15), 2980.

Pramono, P., Irawan, R. D., & Sari, A. A. (2024). Comparative Analysis of SQL Injection Attack Clasification Using Naïve Bayes Method And Support Vector Machine (SVM).

Rajagukguk, N., Kencana, I. P. E. N., & Kusuma, I. G. L. W. (2024). Application of Term Frequency-Inverse Document Frequency in The Naive Bayes Algorithm For ChatGPT User Sentiment Analysis. Proceedings of the First International Conference on Applied Mathematics, Statistics, and Computing (ICAMSAC 2023),

Reddy, S., & Rudra, B. (2021). Evaluation of Recurrent Neural Networks for Detecting Injections in API Requests. 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC),

Reyad, M., Sarhan, A. M., & Arafa, M. (2023). A modified Adam algorithm for deep neural network optimization. *Neural Computing and Applications*, *35*(23), 17095-17112.

Roy, P., Kumar, R., & Rani, P. (2022). SQL injection attack detection by machine learning classifier. 2022 International Conference on Applied Artificial Intelligence and Computing (ICAAIC),

Shah, H. (2024). *SQL Injection dectection using neural network* (Version 1). https://www.kaggle.com/code/waleberhouma/sql-injection-dectection-using-neural-network/input?select=sqli.csv

Sharma, S., Zavarsky, P., & Butakov, S. (2020). Machine learning based intrusion detection system for web-based attacks. 2020 IEEE 6th intl conference on big data security on cloud (BigDataSecurity), IEEE Intl conference on high performance and smart computing,(HPSC) and IEEE Intl conference on intelligent data and security (IDS),

Sherstinsky, A. (2020). Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, *404*, 132306.

Sheykhkanloo, N. M. (2017). A learning-based neural network model for the detection and classification of SQL injection attacks. *International Journal of Cyber Warfare and Terrorism (IJCWT)*, *7*(2), 16-41.

SIAHAAN, E. B., & GIRSANG, A. S. (2023). IMPROVING DETECTION SQL INJECTION ATTACKS USING REFERRER HEADER AND URI WEBLOG. *Journal of Theoretical and Applied Information Technology*, *101*(18).

SQL Injection dectection payloads ((2022). https://pypi.org/project/protego-sqli/#files

Su, G., Wang, F., & Li, Q. (2018). Research on SQL injection vulnerability attack model. 2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS),

Sun, H., Du, Y., & Li, Q. (2023). Deep Learning-Based Detection Technology for SQL Injection Research and Implementation. *Applied Sciences*, *13*(16), 9466.

Sze, V., Chen, Y.-H., Emer, J., Suleiman, A., & Zhang, Z. (2017). Hardware for machine learning: Challenges and opportunities. 2017 IEEE custom integrated circuits conference (CICC),

Tang, P., Qiu, W., Huang, Z., Lian, H., & Liu, G. (2020). Detection of SQL injection based on artificial neural network. *Knowledge-Based Systems*, *190*, 105528.

Xia, Z., Shao, J., Yu, L., Sun, J., Yang, X., Xu, H., Liu, C., & Ren, J. (2024). SQL injection attack detection method based on textCNN. International Conference on Signal Processing and Communication Security (ICSPCS 2024),

Xin, Y., Kong, L., Liu, Z., Chen, Y., Li, Y., Zhu, H., Gao, M., Hou, H., & Wang, C. (2018). Machine learning and deep learning methods for cybersecurity. *Ieee access*, *6*, 35365-35381.

Yacouby, R., & Axman, D. (2020). Probabilistic extension of precision, recall, and f1 score for more thorough evaluation of classification models. Proceedings of the first workshop on evaluation and comparison of NLP systems,

Zhang, W., Li, Y., Li, X., Shao, M., Mi, Y., Zhang, H., & Zhi, G. (2022). Deep Neural Network-Based SQL Injection Detection Method. *Security and Communication Networks*, *2022*(1), 4836289.

Zivkovic, M., Jovanovic, L., Bukumira, M., Antonijevic, M., Mladenovic, D., & Al, M. (2024). Optimizing SQL injection detection using BERT encoding and AdaBoost Classification. Proceedings of the 2nd International Conference on Innovation in Information Technology and Business (ICIITB 2024),

Appendices

All the attached images are in a PDF file and are in this link.

عنوان الرسالة: التحليل المقارن لمجموعة من الخوارزميات لاكتشاف هجمات الحقن في تطبيقات الويب

اسم الطالب الرباعي: أيسر عبد الرحيم مفلح الوشاحي

أسماء لجنة الاشراف:

الدكتور رامي حدرب

الدكتور محمود عبيد

الدكتور أمجد الرطروط

الدكتور رامي يوسف

ملخص

لا تزال هجمات الحقن، وخاصةً حقنSQL ، من أكبر المخاطر التي تهدد أمن التطبيقات عبر الإنترنت، مما يُعرّض خصوصية المستخدم وسلامة النظام وسرية البيانات للخطر. في هذه الرسالة، تُقارن نماذج التعلم الآلي (ML) والتعلم العميق (DL) والنماذج الهجينة للكشف عن مثل هذه الهجمات. بالإضافة إلى النماذج التقليدية مثل الانحدار اللوجستي، وخوارزمية بايز الساذجة، وأشجار القرار، والغابات العشوائية، تم تقييم مجموعة متنوعة من الخوارزميات، بما في ذلك مناهج أكثر تعقيدًا مثل الشبكات العصبية الأمامية(FFNNs) ، والشبكات العصبية التلافيفية(CNNs) ، والتركيبات الهجينة مثل FFNN+Naive Bayes أو.FFNN+SVM

تم استخدام مجموعتي بيانات مختلفتين لتقييم النماذج باستخدام ثلاث تقنيات شائعة لتحويل النصوص إلى متجهاتCount Vectorizer :، وWord2Vec، و .TF-IDFواستُخدمت مقاييس التقييم مثل وقت التنفيذ، والدقة، والدقة، والتذكر، ودرجة F1 لتوفير تقييم شامل. أظهرت النتائج أن شبكات FFNN المضمنة بـ Word2Vec هي نماذج التعلم العميق ذات الدقة الأعلى (تصل إلى 98.9%) وأوقات التنفيذ السريعة نسبيًا. كما أظهرت النماذج الهجينة مثل FFNN + Naive

Bayesو NN + SVMأداءً متميزًا (دقة 98.1%) عند استخدامCount Vectorizer ، جامعةً نقاط قوة كل من التعلم العميق والتعلم الآلي التقليدي.

في المقابل، واجهت النماذج التقليدية مثل Naive Bayes و Decision Treesصعوبة في تمثيل المتجهات الأكثر تعقيدًا أو كثافة، خاصةً مع TF-IDF و .Word2Vecبرز XGBoostكأفضل أداء بين النماذج التقليدية التي تستخدمTF-IDF ، محققًا دقة 98.4% ولكن مع وقت تنفيذ أطول. على الرغم من دقةSVM ، إلا أنها عانت من مشكلات في قابلية التوسع نظرًا لارتفاع تكلفة حسابها في البيانات عالية الأبعاد.

تُظهر هذه الدراسة أنه على الرغم من أن النماذج التقليدية قد تعمل بشكل جيد مع التطبيقات التي تحتاج إلى إكمال سريع، إلا أن تقنيات التعلم العميق والتقنيات الهجينة تتفوق في الحالات التي تكون فيها الدقة مصدر قلق كبير. تشجع النتائج على إجراء المزيد من الأبحاث حول أطر النشر في الوقت الفعلي لاكتشاف حقنSQL ، والتي تدعم أيضًا الاستخدام الأوسع للنماذج الهجينة التي تجمع بين استخراج الميزات الشاملة والتصنيف الفعال.

الكلمات المفتاحية :حقن قواعد البيانات(SQL Injection) ، التعلم الآلي، التعلم العميق، النماذج الهجينة.