



ADAPTIVE CROSS-SITE SCRIPTING DETECTION USING MULTI-AGENT DEEP Q-NETWORK AND HYBRID METHOD FOR TEMPO-SPATIAL DYNAMIC FEATURE SELECTION

By

THAJEEL ISAM KAREEM THAJEEL

**Thesis Submitted to the School of Graduate Studies, Universiti Putra Malaysia,
in Fulfilment of the Requirements for the Degree of Doctor of Philosophy**

July 2024

FK 2024 46

All material contained within the thesis, including without limitation text, logos, icons, photographs and all other artwork, is copyright material of Universiti Putra Malaysia unless otherwise stated. Use may be made of any material contained within the thesis for non-commercial purposes from the copyright holder. Commercial use of material may only be made with the express, prior, written permission of Universiti Putra Malaysia.

Copyright © Universiti Putra Malaysia



DEDICATION

*In the name of Allah, Most Gracious, Most Merciful
This thesis dedicated to:*

My beloved and supportive father, who encouraged me to believe in myself. It was his wish for me to achieve this goal, and I was determined to make it a reality.

My cherished mother, a strong and gentle soul, who taught me to trust in Allah, to believe in hard work, and to understand that much can be accomplished with little.

To my dear brothers, sisters and children (Ali, Abbas, and Baneen), who have supported me throughout this journey. I deeply appreciate everything they have done for me.

And to my dear wife, whose unwavering love and support have been my anchor. Your belief in me has been a source of strength, and I am forever grateful for your presence in my life. This accomplishment is as much yours as it is mine

Last, I dedicate this achievement to my beloved first and second countries, Iraq and Malaysia

Abstract of thesis presented to the Senate of Universiti Putra Malaysia in fulfilment
of the requirement for the degree of Doctor of Philosophy

**ADAPTIVE CROSS-SITE SCRIPTING DETECTION USING MULTI-
AGENT DEEP Q-NETWORK AND HYBRID METHOD FOR TEMPO-
SPATIAL DYNAMIC FEATURE SELECTION**

By

THAJEEL ISAM KAREEM THAJEEL

July 2024

Chairman : Khairulmizam bin Samsudin, PhD
Faculty : Engineering

The ubiquity of modern web applications has deeply integrated them into daily life, but this also exposes them to significant cybersecurity threats, particularly cross-site scripting (XSS) attacks. XSS involves embedding malicious scripts within legitimate webpage code, leading to potential compromise of sensitive information and system integrity. Recent researches have increasingly turned to artificial intelligence (AI) and machine learning (ML) methods to enhance the detection of XSS attacks. However, existing approaches face significant challenges. First, they struggle to address the evolving nature of XSS, where attackers continually refine their tactics to bypass detection systems. This results in temporal feature drift, where the relevance of features changes over time. Second, traditional feature extraction methods often rely on static representations, neglecting the dynamic nature of raw text data, which leads to spatial drift—differences in feature relevance even within the same time period. Consequently, selecting relevant features should be addressed not only at the chunk level of data (as seen in temporal dynamic feature selection) but also at the data group

level for all samples that arrive within a specific time interval. Finally, existing models lack the capacity to handle the diverse types of XSS attacks, which can occur across various data types such as JavaScript, payloads, and statistical features, under different scenarios.

To address these challenges, this research proposes the DQN-MAFS (Deep Q-Network Multi-Agent Feature Selection) model, designed to handle temporal feature drift in the numerical representation of XSS data. In this model, each agent is responsible for selecting or deselecting a specific feature, with rewards distributed fairly through the Fair Agent Reward Distribution (FARD-DFS) mechanism. Additionally, the spatial and temporal feature drifts in raw text XSS data are tackled using the Tabu Search-Based Tempo-Spatial Dynamic Feature Selection (2TS-DFS). This model employs dynamic word embedding method that consist of Word2Vec and Bi-LSTM models. To handle the issue of model capacity, both DQN-MAFS and 2TS-DFS are integrated into a unified framework capable of processing two types of XSS data: numerical and text-based (including JavaScript and payloads).

The proposed models were evaluated on six distinct XSS datasets: four numerical datasets (D1-N66, D2-N167, D3-N30, and D4-N30) and two raw text datasets (D5-JS, a JavaScript dataset, and D6-PL, a payload dataset). Furthermore, the proposed models outperformed benchmark methods designed for XSS attack detection and dynamic feature selection, including those utilizing reinforcement learning, genetic algorithms, multi-agent reinforcement learning, JSContana and static word embedding techniques, in terms of accuracy, precision, recall, and F1-score. The DQN-MAFS and 2TS-DFS models demonstrated superior performance, achieving a mean accuracy of 99.59% and

an F1-score of 98.42%. The results affirm the framework's efficacy in enhancing cybersecurity measures by providing a scalable, adaptable, and efficient adaptive approach to the evolving nature of XSS attack.

Keywords: Cyber Security, Dynamic Feature Selection, Feature Drift, Machine Learning, Web Application Security.

SDG: GOAL 9: Industry, Innovation and Infrastructure



Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia sebagai memenuhi keperluan untuk ijazah Doktor Falsafah

**PENGESANAN CROSS-SITE SCRIPTING ADAPTIF MENGGUNAKAN
JARINGAN Q MENDALAM MULTI-EJEN DAN KAEDAH HIBRID UNTUK
PEMILIHAN CIRI DINAMIK TEMPO-SPASIAL**

Oleh

THAJEEL ISAM KAREEM THAJEEL

Julai 2024

Pengerusi : Khairulmizam bin Samsudin, PhD
Fakulti : Kejuruteraan

Kewujudan aplikasi web moden yang semakin meluas telah menjadikannya sebahagian daripada kehidupan harian kita, namun ini juga mendedahkannya kepada ancaman keselamatan siber yang ketara, terutamanya serangan cross-site scripting (XSS). XSS melibatkan penyisipan skrip berbahaya ke dalam kod halaman web yang sah, yang boleh menyebabkan kompromi terhadap maklumat sensitif dan integriti sistem. Kajian terkini semakin menggunakan kaedah kecerdasan buatan (AI) dan pembelajaran mesin (ML) untuk meningkatkan pengesanan serangan XSS. Namun, pendekatan yang sedia ada menghadapi cabaran yang ketara. Pertama, mereka menghadapi kesukaran untuk menangani sifat XSS yang sentiasa berkembang, di mana penyerang sentiasa memperhalusi taktik mereka untuk melepasi sistem pengesanan. Ini menyebabkan peralihan ciri temporal, di mana kepentingan ciri berubah dari masa ke masa. Kedua, kaedah pengekstrakan ciri tradisional sering bergantung kepada representasi statik, mengabaikan sifat dinamik data teks mentah, yang membawa kepada peralihan ciri spatial—perbezaan dalam kepentingan ciri

walaupun dalam tempoh masa yang sama. Oleh itu, pemilihan ciri yang relevan perlu ditangani bukan sahaja pada tahap kelompok data (seperti yang dilihat dalam pemilihan ciri dinamik temporal) tetapi juga pada tahap kumpulan data untuk semua sampel yang tiba dalam selang masa tertentu. Akhirnya, model yang sedia ada kekurangan kapasiti untuk menangani pelbagai jenis serangan XSS, yang boleh berlaku merentasi pelbagai jenis data seperti JavaScript, payload, dan ciri statistik, dalam pelbagai senario.

Untuk menangani cabaran-cabaran ini, kajian ini mencadangkan model DQN-MAFS (Pemilihan Ciri Berbilang Ejen Berasaskan Rangkaian Q Mendalam), yang direka untuk menangani peralihan ciri temporal dalam representasi berangka data XSS. Dalam model ini, setiap ejen bertanggungjawab untuk memilih atau tidak memilih ciri tertentu, dengan ganjaran diagihkan secara adil melalui mekanisme Pengagihan Ganjaran Ejen Adil Berasaskan Pemilihan Ciri Dinamik (FARD-DFS). Selain itu, peralihan ciri spatial dan temporal dalam data teks mentah XSS ditangani menggunakan Pemilihan Ciri Dinamik Tempo-Ruang Berasaskan Pencarian Tabu (2TS-DFS). Model ini menggunakan kaedah embedding perkataan dinamik yang terdiri daripada model Word2Vec dan Bi-LSTM. Untuk menangani isu kapasiti model, kedua-dua DQN-MAFS dan 2TS-DFS diintegrasikan ke dalam kerangka kerja bersatu yang mampu memproses dua jenis data XSS: berangka dan berasaskan teks (termasuk JavaScript dan payload).

Model yang dicadangkan telah dinilai menggunakan enam set data XSS yang berbeza: empat set data berangka (D1-N66, D2-N167, D3-N30, dan D4-N30) dan dua set data teks mentah (D5-JS, satu set data JavaScript, dan D6-PL, satu set data payload).

Selanjutnya, model-model yang dicadangkan telah mengatasi kaedah-kaedah rujukan yang direka untuk pengesanan serangan XSS dan pemilihan ciri dinamik, termasuk yang menggunakan pembelajaran pengukuhan, algoritma genetik, pembelajaran pengukuhan berbilang ejen, JSContana, dan teknik penyematam kata statik, dari segi ketepatan, kepresisan, ingatan, dan skor F1. Model DQN-MAFS dan 2TS-DFS menunjukkan prestasi unggul, mencapai ketepatan purata 99.59% dan skor F1 sebanyak 98.42%. Keputusan ini mengesahkan keberkesanan kerangka kerja dalam meningkatkan langkah keselamatan siber dengan menyediakan pendekatan yang boleh diskalakan, mudah disesuaikan, dan cekap terhadap sifat serangan XSS yang sentiasa berkembang.

Kata kunci: Ciri Dinamik, Keselamatan Siber, Ciri, Pembelajaran Mesin, Pemilihan Hanyutan Keselamatan Aplikasi Web.

SDG: MATLAMAT 4: Industri, Inovasi dan Infrastruktur.

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support of many different people. First and foremost, solemn praise and humble gratitude “to Allah, the Almighty, for bestowing His blessings on me throughout the research work to its successful completion.

I would like to thank my supervisor Dr. Khairulmizam Bin Samsudin, who introduced me to web security and mentored me ever since. He taught me how to find and develop interesting ideas and transform these ideas into proper research papers. Without his support, this thesis would not exist today. His dynamism, vision, sincerity, and motivation have deeply inspired me. He has taught me the methodology to carry out the research and present the research works as clearly as possible. It was a great privilege and honor to work and study under his guidance. I am incredibly grateful for what he has offered me. I would also like to thank him for his friendship, empathy, and great sense of humor.

Furthermore, I would like to thank the rest of supervision committee: Assoc. Prof. Dr. Fazirulhisyam Bin Hashim and Prof. Dr. Shaiful Jahari Bin Hashim for supervising and helping me during my Ph. D journey.

Last but not least, I am incredibly grateful to my parents, wife, kids and brothers for their care, love, prayers, support, patience and valuable sacrifices to educate and prepare me for my future.

This thesis was submitted to the Senate of Universiti Putra Malaysia and has been accepted as fulfilment of the requirement for the degree of Doctor of Philosophy. The members of the Supervisory Committee were as follows:

Khairulmizam bin Samsudin, PhD

Senior Lecturer
Faculty of Engineering
Universiti Putra Malaysia
(Chairman)

Shaiful Jahari bin Hashim, PhD

Professor, Gs.
Faculty of Engineering
Universiti Putra Malaysia
(Member)

Fazirulhisyam bin Hashim, PhD

Associate Professor
Faculty of Engineering
Universiti Putra Malaysia
(Member)

ZALILAH MOHD SHARIFF, PhD

Professor and Dean
School of Graduate Studies
Universiti Putra Malaysia

Date: 10 October 2024

TABLE OF CONTENTS

	Page
ABSTRACT	i
ABSTRAK	iv
ACKNOWLEDGEMENTS	vii
APPROVAL	viii
DECLARATION	x
LIST OF TABLES	xv
LIST OF FIGURES	xvii
LIST OF ABBREVIATIONS	xix
 CHAPTER	
 1 INTRODUCTION	 1
1.1 Introduction	1
1.2 Research Problem	6
1.3 Research Significance	11
1.4 Research Objectives	12
1.5 Research Contributions	12
1.6 Research Scope	13
1.7 Thesis Organization	15
 2 LITERATURE REVIEW	 18
2.1 Introduction	18
2.2 Cyber Security in The Web Application Domain	21
2.3 Cross-site Scripting Attacks (XSS)	26
2.3.1 XSS Types	27
2.3.2 XSS Attack Forms and Components	31
2.3.3 XSS Evolution	35
2.4 Classical XSS Attack Detection Methods	38
2.4.1 Static Analysis	39
2.4.2 Dynamic Analysis	40
2.4.3 Hybrid Analysis	40
2.4.4 Web Content Filters	41
2.5 Data Stream Classification	42
2.5.1 Concept Drift	43
2.5.2 Feature drift	44
2.6 Feature Selection Methods	47
2.6.1 Selection-based Knowledge Methods	48
2.6.2 Filtering Methods	48
2.6.3 Wrapper Methods	49
2.6.4 Embedded Methods	50
2.6.5 Attention Models	50
2.7 Related Works	53
2.7.1 ML/DL-based XSS Attack Detection Approaches	54
2.7.2 RL For Dynamic Feature Selection (DFS)	77

2.8	Open Challenges and Gap Discussion	83
2.9	Preliminary Background of the Proposed Models' implementation	85
2.9.1	Decision Trees (DTs)	86
2.9.2	Support Vector Machine (SVM)	89
2.9.3	Agglomerative Clustering	92
2.9.4	Reinforcement Learning (RL)	95
2.9.5	Deep learning approaches	105
2.9.6	Tabu Search Overview	115
2.10	Chapter summary	117
3	RESEARCH METHODOLOGY	118
3.1	Introduction	118
3.2	Methodology Design	119
3.3	Problem Formulation	123
3.4	DA-XSS-AD Framework Overview	124
3.5	Considerations and Conceptual Basis of the DA-XSS-AD Framework	127
4	IMPLEMENTATION OF DA-XSS-AD'S MODES	131
4.1	Introduction	131
4.2	DQN-based Multi-Agent Features Selection (DQN-MAFS)	132
4.2.1	Preparation of Data Chunking	133
4.2.2	DQN-MAFS Model	134
4.2.3	General DQN-MAFS Algorithm	143
4.2.4	Detector	146
4.3	Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS)	146
4.3.1	Data Decomposition	148
4.3.2	Preprocessing	149
4.3.3	Adaptable Context Analysis	153
4.3.4	Dynamic Feature Selection	162
4.3.5	Classification	170
4.3.6	2TS-DFS Algorithm	171
4.4	Evaluation Methodology	174
4.4.1	Evaluation Metrics	174
4.4.2	Statistical analysis	177
4.4.3	Datasets	179
4.4.4	Experimental Design	185
4.5	Chapter Summary	193
5	RESULTS AND DISCUSSION	195
5.1	Introduction	195
5.2	DQN-MAFS Model	196
5.2.1	Evaluation of DQN-MAFS With Different Classifiers	196
5.2.2	Comparative Analysis	202
5.2.3	Discussion	212
5.3	2TS-DFS Model	217
5.3.1	Evaluation of 2TS-DFS Model	218
5.3.2	Comparative Analysis	232

5.3.3	Statistical significance analysis	243
5.3.4	Memory Consumption	247
5.3.5	Comparisons with other feature selection methods	257
5.3.6	Discussion	263
5.4	DA-XSS-AD Framework	269
5.4.1	Evaluation of DA-XSS-AD	270
5.4.2	Feature Reduction	275
5.5	Overall Discussion	279
5.6	Chapter Summary	282
6	CONCLUSION AND FUTURE WORKS	283
6.1	Conclusion	283
6.2	Future Works	287
REFERENCES		289
BIODATA OF STUDENT		308
LIST OF PUBLICATIONS		309

LIST OF TABLES

Table	Page
2.1 Payload-based XSS components with examples	32
2.2 After and before applying the encoding methods	36
2.3 Literature Survey of the existing ML -based XSS detection approaches	62
2.4 Literature Survey of the existing works in XSS detection using deep learning models	74
2.5 Literature Survey of the existing algorithms in RL based dynamic feature selection	82
4.1 Decoding example for payload and JS malicious files	150
4.2 Regularized word segmentation rules for payload	151
4.3 An example of extracted syntactic units from the AST	156
4.4 Confusion Matrix parameters	174
4.5 Existing numerical XSS benchmarking datasets	182
4.6 Existing raw text-based benchmarking datasets	184
4.7 Hardware details of the implementation environment	185
4.8 Chunk composition and splitting	188
4.9 Parameters of the DQN-MAFS model	188
4.10 Parameters of the 2TS-DFS model	191
4.11 Chunk composition and splitting	192
5.1 Numerical comparison of our developed FARD-DFS with Tariq et al., (2021) and K. Liu et al., (2021) by using D1-N66 dataset	205
5.2 Numerical comparison of our developed FARD-DFS with Tariq et al., (2021) and K. Liu et al., (2021) by using D2-N167 dataset	207
5.3 Numerical comparison of our developed FARD-DFS with Tariq et al., (2021) and K. Liu et al., (2021) by using D3-N30 dataset	209
5.4 Numerical comparison of our developed FARD-DFS approaches with Tariq et al., (2021) and K. Liu et al., (2021) by using D4-N30 dataset	211

5.5	Overall accuracy results of the methods with respect to different feature size datasets	213
5.6	Numerical comparison at data chunk level of our developed FARD-DFS with Tariq et al., (2021) and K. Liu et al., (2021) by using D2-N167 dataset	216
5.7	t-test statistical significance testing of the accuracy and F1-measure results in comparison with benchmarks methods	217
5.8	t-test statistical significance testing of the accuracy and F1-measure results in comparison with benchmarks methods	244
5.9	Shapiro-Wilk Test statistical significance testing of the accuracy results in comparison with benchmarks methods	246
5.10	Comparison of 2TS-DFS with other feature selection methods using D5-JS datasets	260
5.11	Comparison of 2TS-DFS with other feature selection methods using D6-PL datasets	262
5.12	Overall comparison of 2TS-DFS model by using D5-JS and D6-PL datasets	266
5.13	Classification accuracy of DA-XSS-AD framework and sub-models in comparison with other baseline approaches using different input data types and representations	272
5.14	F1-scores of DA-XSS-AD framework and sub-models in comparison with other baseline approaches using different input data types and representations	274
5.15	Feature reduction rate (FRR) of the DA-XSS-AD framework and sub-models in comparison with other approaches	278

LIST OF FIGURES

Figure	Page
2.1 Literature Review Map	20
2.2 OWASP high risk vulnerability statistics and new categories	25
2.3 Reflected XSS attack steps	28
2.4 Stored XSS attack steps	29
2.5 DOM-based XSS attack steps	30
2.6 Attack vector payload sample with components' labels correspond to Table 2.1	32
2.7 Main types of feature drift	46
2.8 Basic structure of Reinforcement learning algorithm	96
2.9 Convolution Neural Network architecture	106
2.10 Long short-term cell structure	109
2.11 Structure of the Bi-LSTM network	112
3.1 Methodology design overview	122
3.2 DA-XSS-AD framework	127
4.1 The overall DQN-MAFS model	133
4.2 Our proposed sub-model for DQN-MAFS, which is used for fair reward distribution to agents and is designated as fair agent reward distribution based dynamic feature selection FARD-DFS	138
4.3 Reward calculation for ACC and ALC approaches	140
4.4 Overall 2TS-DFSmodel	147
4.5 An Example of generating an Abstract Syntax Tree (AST)	152
4.6 Dynamic word embedding approach overview	154
4.7 Skip-gram and CBOW models	157
4.8 Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS)	169

5.1	Evaluation of DQN-MAFS and its FARD-DFS approaches with different classifiers in terms of accuracy, precision, recall, and f1-measure metrics, by using D2-N167 datasets	201
5.2	Training and testing results of our proposed 2TS-DFS model using D5-JS dataset	224
5.3	Training and Testing results of the proposed 2TS-DFS model using D6-PL dataset	231
5.4	Comparison of our proposed 2TS-DFS model with others in terms of classification accuracy, precision, recall, f1-score, and false positive metrics, by using D5-JS dataset	237
5.5	Comparison of our proposed 2TS-DFS model with others in terms of classification accuracy, precision, recall, f1-score, and false positive metrics, by using D6-PL dataset	242
5.6	Selected features of 2TS-DFS over clusters for each chunk using D5-JS dataset	250
5.7	Selected features of 2TS-DFS over clusters for each chunk using D5-JS dataset	254
5.8	Selected features of 2TS-DFS over chunks in comparison with others using D5-JS and D6-PL datasets	257
5.9	Prediction time of 2TS-DFS model over chunks in comparison with others using D5-JS and D6-PL datasets	268

LIST OF ABBREVIATIONS

2TS-DFS	Tabu Search based Tempo-spatial Dynamic Feature Selection Model
ACC	Accumulative Contribution
Adaboost	Adaptive Boosting
AI	Artificial Intelligence
AL	Aspiration Criteria or Level
ALC	Alternative Contribution
ANNs	Artificial Neural Networks
API	Application Programming Interface
AST	Abstract Syntax Tree Representation
BFE	Backward Feature Elimination
Bi-LSTM	Bi-directional Long Short-Term Memory
Bi-RNN	Bi-directional Recurrent Neural Network
BoW	Bag-of-Words
CBoW	Continuous Bag-of-Words
CHI	Chi-square test
CI/CD	Continuous Integration (CI)/Continuous Deployment (or Delivery)
CM	Combination Matrix
CNN	Convolutional Neural Network
Code2Vec	Code-to-vector model
CSP	content security policies
CSS	Cascading Style Sheets
CTDE	centralized training with decentralized execution
CVE	Common Vulnerabilities and Exposures Program
DBN	Deep Belief Network

DDOS	Distributed Denial of Service
DDQN	double-deep Q-network
DF	document frequency
DFS	Dynamic feature selection
DL	Deep Learning
DNN	Deep Neural Network
Doc2Vec	Document-to-vector model
DOM	Document Object Model
DoS	Denial of Service
DQN	Deep Q-Network
DQN-MAFS	Deep Q-Network Multi Agent Feature Selection Model
DT	Decision Trees
EL	Expression Language
FARD-DFS	fair agent reward distribution based dynamic feature selection
FC	Fully Connected Layer
FFS	Forward Feature Selection
FPR	False Positive Rate
FRR	Feature reduction ratio
GAN	Generative Adversarial Network
GBoost	Gradient Boosting
GCN	Graphic Convolutional Network
GRU	Gated Recurrent Unit
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IG	Information Gain
IM	Impurity Measurement
JS	JavaScript code

JSON	JavaScript Object Notation file
KNN	k-Nearest Neighbors
LR	Logistic Regression
LSTM	Long Short-Term Memory
MARL	Multi-Agent RL
MDP	Markov Decision Process
MG	Markov Game
ML	Machine Learning
MLP	multilayer perceptron
MOEFS	Multi-Objective Evolutionary Feature Selection
NB	Naïve Bayesian
NLP	Natural Language Processing
NLTK	Nature Language Toolkit Module
OA-OT	One Action at One Time
OGNL	Object Graph Navigation Library
ORM	Object Relational Mapping ORM
OWASP	Open Web Application Security Project
PLP	Programming Language Processing
RBF	Radial Basis Function
RF	Random Forest
RFE	Recursive Feature Elimination
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SBS	Sequential backward selection
SM	Softmax Function
SQL	SQL injection attack
SSRF	Server-Side Request Forgery

SVM	Support Vector Machine
TL	Tabu List
TS	Tabu Search algorithm
UCB	Upper Confidence Bound
URL	Uniform Resource Locator
WAF	Web application firewall
Word2Vec	Word-to-vector model
XGBoost	Extreme Gradient Boosting algorithm
XSS	Cross-site scripting attack
ϵ -greedy	Epsilon-Greedy strategy

CHAPTER 1

INTRODUCTION

1.1 Introduction

In modern days, people are surrounded by billions of web application pages in many distinct domains; with just a single click by the user, enormous amounts of data streams are produced at extremely high-speed rates. Most of these web applications are high-frequency use applications that impose fast-evolving data to respond immediately and dynamically (X. Wang et al., 2022). In this case, the data is processed in real-time and the browser environment is changed due to the client interactions. At the same time, cyber threats are evolving to conduct attacks with high severity levels (Sun et al., 2019). Regretfully, web applications are vulnerable to various cyber-attacks, and the consequences of a successful web application attack can be severe, including financial losses, reputational damage, and legal liabilities.

According to Open Web Application Security Project (OWASP) (OWASP, 2017, 2021), the top 10 web application attacks are: (1) broken access control, (2) cryptographic failures, (3) injection attacks (including Cross-site scripting (XSS), SQL injection, and command injection attacks), (4) insecure design, (5) security misconfiguration, (6) vulnerable and outdated components, (7) identification and authentication failures, (8) software and data integrity failures, (9) security logging and monitoring failures, and (10) server-side request forgery (SSRF). One of the most common forms of evolving cyberattacks that threat the web application security as an

injection attack is Cross-site scripting (XSS) (C. Gupta et al., 2022; Hannousse et al., 2024; Sarmah et al., 2020; Tariq et al., 2021; Y. Zhou & Wang, 2019). According to a ranking published by the Open Web Application Security Project (OWASP) (OWASP, 2017), XSS attacks respectively ranked third and seventh in global web application security risks in 2013 and 2017. In the newly report released in 2021(OWASP, 2021), XSS is classified as a security risk in the “Injection” category, ranking third. These data show that XSS attacks are still a serious and most frequent threat to user security and privacy. Moreover, the total XSS attacks have sharply risen over the years. Currently, the XSS has the third-highest total count with 23,363 attacks from its discovery time to March 2023, making approximately 12.7 % of the total reported vulnerabilities (SecurityScorecard., n.d.).

The attacker can conduct the XSS attack by exploiting the vulnerabilities that may be found in Web applications due to the use of improper or dangerous scripts by developers (Sarmah et al., 2018). An XSS attack is used to threaten various domains such as individual privacy, medical, economics, and even the military (Rodriguez et al., 2020). Typically, attackers employ different strategies to execute XSS attacks, irrespective of the targeted domain. The primary distinction lies in the type of information they aim to expose, depending on the specific XSS variant—be it stored, reflected, or DOM-based. At the initial step, the attacker looks for and identifies potential vulnerabilities in the web application (Z. Liu et al., 2023b). A malicious script (payload) is created and injected into a susceptible Web application's trusted context as an embedded code or as JavaScript file. The URL link of this web application is given to the victim via various media. When the victim is lured to click on the received link, the browser executes the malicious script that served as part of the legitimate code

because of the browser's inability to distinguish between malicious and normal scripts. As a result, the users of vulnerable applications fall prey to attackers who obtain sensitive information.

Various classes of methods were used to counter XSS. Initially, with the emergence of XSS attacks to the public, specialists relied on traditional methods to detect the XSS attacks on the client-side, server-side, or pair sides. The traditional approaches can be broadly classified into three kinds: static, dynamic, and hybrid analysis (Mereani & Howe, 2018; Rodriguez et al., 2020). The static analysis method focuses on reviewing the source code without running it, while the dynamic analysis approach focuses on analysing the behaviour of the script during execution with some input. Finally, Hybrid analysis combines static and dynamic analysis to mitigate the disadvantages of both techniques. Static, dynamic, and hybrid are regarded as classical XSS defence methods. Such traditional methods might provide some promises that such XSS vulnerabilities will not occur (Kaur et al., 2023). However, they are also criticised for being slow, unable to provide an efficient result due to its high false positive rates, complex, costly in terms of both resource-consuming and time-consuming, and requiring high-security domain knowledge (M. Liu et al., 2019; R. Wang et al., 2018b). Current researchers attempt to leverage artificial intelligence (AI) principles to add intelligence to boost detection, as seen by the widespread usage of machine learning/deep learning (ML/DL) techniques in detecting cyber-attacks. On the other hand, traditional machine-learning techniques have a low detection rate and are incapable of detecting minor mutants of current malicious attacks, such as unknown/new variant attacks (X. Li et al., 2023; Mokbal et al., 2022; Sahoo et al., 2019). Moreover, the evolving nature of attacks requires an ML/DL-based approach

equipped with dynamic awareness. More specifically, the monitoring of drift occurrence and adapting the model accordingly. One special type of drift that researchers in web security applications have ignored is feature drift, which indicates the dynamic or temporal change in the relevancy of features. Such change leads to a model trained with part of non-relevant features, and consequently, a degradation in the accuracy occurs (Barddal et al., 2017; Garcia et al., 2023; Sato et al., 2022). To handle such issues, researchers relied on traditional feature selection methods to select the most relevant features. The traditional feature selection methods have a static nature and are based on batch learning which assumes that the set of relevant features does not change over time and needs multiple passes over data. However, in many applications, and XSS is one of them, the features might be subject to change in their relevancy with respect to time due to the evolving nature of attacks. The evolved XSS attack can be defined as confusing and aggressive innovative XSS attacks that are generated with different combinations of features from the benign or malicious existing XSS records (C. Gupta et al., 2022; X. Li et al., 2023). Usually, such combinations are created by using evading strategies such as encoding, obfuscation techniques, replacing the alphabet style lowercase/uppercase, removing, or substituting the JavaScript and HTML attributes/functions/events/tags/symbols/expression, adding or removing the blank spaces, etc. Security experts (OWASP, XSS cheat sheet) (Jim Manico, 2018) and researchers (Stock et al., 2017) have highlighted and validated these various combinations through extensive analysis of different repositories.

Feature drift has emerged only recently, and a limited number of techniques have been developed so far (Barddal et al., 2017; Garcia et al., 2024; X. Wang et al., 2022).

Dynamic feature selection (DFS) is an emerging sub-topic in machine learning that has been used recently to capture feature drift (Barddal et al., 2016; Ferone & Maratea, 2021; Villa-Blanco et al., 2023). It involves selecting an optimal subset of features based on their relevancy and filtering out non-relevant features with respect to a set of records. DFS can be applied to both stream data and non-stream data. However, the DFS is more convenient for the stream and sequential nature of data due to the drift that indicates changing the distribution that represents the source of the data. In addition, it is useful for non-temporal data when there is diversity in data structure which can be termed as cluster level feature selection or spatial feature selection. The terms "spatial" and "temporal" pertain to the multi-dimensional attributes of data features, rather than their geographical or physical locations. Here, "spatial" refers to the arrangement or relationship of features within a given data structure, which could be a neural network, a matrix, or any other form of organized data. "Temporal," on the other hand, relates to how these features evolve or change over discrete time steps, such as in time-series data or sequential algorithms. This is distinct from the usage of "spatial" and "temporal" in geographical or physical sciences, where "spatial" typically refers to the geographical location or physical arrangement of objects in the real world, and "temporal" denotes the change in these objects or phenomena over real-world time (Hamdi et al., 2022; S. Wang et al., 2022).

The primary goal of this thesis is to create a robust and efficient method for detecting and countering evolving XSS attacks, leveraging Dynamic Feature Selection (DFS) in combination with advanced Machine Learning/Deep Learning techniques. The proposed methodology will effectively consider both temporal and spatial aspects of feature selection to capture the evolving nature of XSS attacks and improve detection

accuracy. Temporal feature selection will cater to the need of identifying changing relevancy of features over time due to the dynamically evolving nature of attacks, while spatial feature selection will be utilized to capture the diversity in data structure and hence improve the adaptability of the model to varied attack patterns. This comprehensive approach is expected to recognize and respond effectively to feature drift, a critical and largely overlooked aspect of evolving cyberattacks in web security applications. The approach is also expected to handle a diversity of input data that caused the XSS attack within different scenarios to show its capacity and adaptability in detecting such attacks.

1.2 Research Problem

Web applications have become increasingly pervasive, serving as critical platforms for data exchange and user interaction (Q. Wang et al., 2022). Their widespread use has also made them prime targets for malicious actors who frequently inject harmful scripts into legitimate web pages. These attackers use sophisticated evading strategies to evade traditional detection systems, contributing to a fast-evolving threat landscape. However, most research in this domain is rooted in artificial intelligence and machine learning principles aimed at bolstering detection capabilities (Z. Liu et al., 2023a; Mokbal et al., 2021; Sahoo et al., 2019). Despite these advancements, existing approaches often fall short in handling the diversity of data types and representation, evolving nature of attacks, focusing primarily on offline learning and ignoring incremental learning without providing for real-time adaptation and updates (Alazab et al., 2022; C. Gupta et al., 2022; Lekies, 2023; Tariq et al., 2021). Thus, the evolving nature of attacks requires an ML/DL-based approach support incremental learning and

equipped with dynamic awareness for monitoring of drift occurrence and adapting the model accordingly.

While previous studies have emphasized the importance of feature extraction and selection, there is a glaring gap in the literature concerning the dynamic nature of feature relevancy, often referred to as "feature drift" (Barddal et al., 2017; Garcia et al., 2023; Sato et al., 2022). This neglect of dynamic feature selection has repercussions for model accuracy over time. Current literature relied on traditional feature selection methods to select the most relevant features. The traditional feature selection methods have a static nature and are based on batch learning which assumes that the set of relevant features does not change over time and needs multiple passes over data (K. Liu et al., 2021; Villa-Blanco et al., 2023). In such scenarios, feature selection occurs in a static manner, where a fixed number of features is selected for the entire dataset during the offline training phase (Barddal et al., 2019; Fahy & Yang, 2019; X. Wang et al., 2022). However, in many applications, and XSS is one of them, the features might be subject to change in their relevancy with respect to time due to the evolving nature of attacks. To tackle this challenge, the literature has introduced a Reinforcement Learning (RL)-based approach for dynamic feature selection (DFS).

The current state of RL-based DFS methods, as discussed in Chapter 2 (Section 2.7.2 and Table 2.5), reveals significant challenges and limitations. Existing RL-based DFS methods can be broadly classified into two categories: Single-agent DFS and Multi-agent DFS. Single-agent DFS methods, such as those proposed by (Y. Cheng et al., 2021; Covert et al., 2023; X. Li et al., 2021; Wu et al., 2022; R. Xu et al., 2021), involve a single RL agent responsible for selecting all features. While this design is computationally efficient due to the optimization of a single policy network, it

struggles as the feature space expands. The action space increases exponentially with the number of features (K), leading to the potential for suboptimal, locally confined solutions. Conversely, Multi-agent DFS methods, exemplified by works such as (Fan et al., 2020, 2021; Z. Fang et al., 2019; M. Kim et al., 2022; K. Liu et al., 2021; Paniri et al., 2021; Ren et al., 2023), deploy multiple agents, each tasked with selecting specific features. However, from the perspectives of the elements of RL, namely, state, action, reward, agent type, and exploration vs. exploitation balance, it is observed that non-of the existing Multi-agent DFS methods have provided an algorithm for reward distribution to handle the multi-agent rewarding. The tricky part of multi-agent rewarding is that the agents' contribution to the overall performance is not equal (Gronauer & Diepold, 2022; T. T. Nguyen et al., 2020). This is because each agent's decision is weighted based on the importance of its feature in the overall performance; figuring out this importance is not a straightforward process. Hence, it is necessary to come up with an approach that accomplishes fair reward distribution based on the real contribution of each agent. Such approach is useful for dealing with temporal type of drifts when features are represented in their simplest way as numerical features. Hence, the framework is RL is useful only when we have numerical representation of XSS features.

Current approaches in XSS attack detection predominantly depend on either static or dynamic word embeddings for text-based feature extraction (Y. Huang et al., 2021; Mokbal et al., 2022; Y. Wang et al., 2020). While these methods may be adopted at general feature representation, they are deficient in handling the evolving nature of attacks effectively. Specifically, they fail to select the most relevant portions of the hidden state representation that capture the essence of the evolving threats (Garcia et

al., 2023, 2024). The only work that has used dynamic word embeddings is the work of (Y. Huang et al., 2021). This work is criticized by lacking important sub-functionalities that improve the performance. First, it does not consider the high dimension of extracted features which might affect efficiency. In order to overcome this, an additional stage of feature selection is needed. Second, the real operation of XSS detection model is stream based which means that data comes sequentially. However, this was not considered by all the previous works which means that no attention was made to concept drift nor feature evolving. With these types of features, the handling of the evolving nature of data should consider two types of feature drift. The first one is temporal which assumes that certain type of features might not be active in the samples that are indexed with certain time index or belongs to the same chunk. The second one is spatial which assumes that certain types of features might be relevant in sub-set of samples and non-relevant in other sub-set even when both sub-sets are indexed with the same time step or belonging to the same chunk. The second type necessitate enabling instance level feature selection in order to handle both types instead of chunk level only. This would allow for more precise selection and adaptation of features based on the specific characteristics of each individual sample.

Unlike numerical data, where temporal changes are the primary concern, text-based data require a more intricate selection process. The decision of what features to select should be twofold: it should account for temporal changes in the data and also consider the spatial similarity among various records within a data chunk. This nuanced approach enables group-level or cluster-level feature selection, making the detection system both temporally and spatially aware. Tabu search, known for its local searching capabilities, stands as an apt candidate to tackle this multifaceted problem, offering the

potential to combine both temporal and spatial awareness effectively. One of the limitations of Tabu search is the risk of oscillation between two solutions due to the limited length of Tabu list (Cavero et al., 2024). However, this issue is avoided in our solution because of the high dimensionality of the solution which decreases the probability of this phenomena.

Furthermore, an issue with the existing works on XSS is that they are evaluated on one type of XSS data or one specific data representation. Some researchers evaluated their approaches solely on payloads data (Kuppa et al., 2022), (Akaishi et al., 2019), (Maurel et al., 2022), (Lee et al., 2022), (Abaimov & Bianchi, 2019), (Chaudhary et al., 2021); while other solely on JavaScript files (Alazab et al., 2022), (Y. Huang et al., 2021), (Rozi et al., 2022). Additionally, some researchers evaluate on numerical dataset extracted from the payload (Mokbal, Dan, Imran, Lin, et al., 2019), (Mokbal et al., 2021), (Y. Zhou & Wang, 2019), (Tariq et al., 2021). This limits scope of evaluation which makes the assessment incomplete and may not reflect the real-world XSS detection scenarios accurately. The numerical dataset is a dataset that its features have been extracted by using statistical feature extraction methods.

Overall, Web applications are vital, yet vulnerable platforms frequently targeted by sophisticated cyber-attacks. Despite AI-driven advancements in cyber-security, existing approaches lack real-time adaptability and struggle with diverse data representations. Current research neglects the dynamic nature of feature relevancy, or "feature drift," compromising model accuracy over time. Multi-agent Reinforcement Learning is proposed for dynamic numerical feature selection, but text-based data present more complex requirements. For textual data, existing methods using static or

dynamic word embeddings fail to adapt to evolving threats effectively. Tabu search combined with dynamic word embedding offer a promising solution, uniquely capable of addressing both temporal and spatial drifts in feature selection. In contrast to previous researchers, this research aims to overcome the limitation of handling only a single data type or representation by proposing a framework capable of detecting XSS attacks using two distinct types of data and representations: raw textual data (e.g., JavaScript and payload) and numerical data extracted from the payload. By integrating multiple data types and representations, the proposed framework enhances the capacity to detect XSS attacks across a wide range of

1.3 Research Significance

This research presents a deeper insight of cross-site scripting attacks and the adopted machine/deep learning approaches in the web application security domain. In addition, by obtaining the research aim and objectives, this work will hopefully benefit the web application users and developers to protect them against such attacks on the both sides, client-side, and server-side. As the proposed approach can be deployed as an autonomous detection system without the need for any offline retraining process of the model to detect the evolved XSS attack. Instead, the model will track the drift caused by feature evolving and adapt to the temporal and spatial changes. Furthermore, the model will enable prediction on a reduced number of high-level features which means a memory reduction. In addition, the proposed model has a capacity to deal with a variety type of malicious data and different data representation. For instance, the model can be learned with Payload and JS file type, also can be learned with numerical representation extracted by statistical-based feature extraction methods, or dynamic

hidden states representation that are extracted using dynamic word embedding method from raw textual data. The implications of our research are significant, as the proposed methodology provides an efficient and effective solution for detecting evolving XSS attacks, enhancing web application security. Its adaptability, performance, and memory efficiency make it a valuable tool for developers and security professionals.

1.4 Research Objectives

The research aims at proposing an XSS attack detection approach that handles data evolving. The aim is accomplished by the following objectives:

1. To propose a unified framework for cross-site scripting that handles diverse types of data and supports incremental and evolving nature of learning.
2. To develop RL based temporal dynamic feature selection for numerical representation of cross-site scripting attacks detection.
3. To develop Tabu search based temporal and spatial dynamic feature selection using hidden state representation of JS and Payload data.
4. To evaluate the effectiveness and efficiency of the proposed framework's models to compare with benchmark approaches.

1.5 Research Contributions

This thesis achieved several main contributions are listed as follows:

1. It proposes a unified framework for XSS detection that includes various functionalities, namely, pre-processing, adaptable context analysis, and a spatial-temporal dynamic feature selection. The latter is the first in the domain of XSS detection that considers incremental learning and enables knowledge updates for both feature selection and classification at the same time.
2. It proposes multi-agent temporal dynamic feature selection using RL by including several components. The framework is named as deep Q-network multi agent feature selection DQN-MAFS. Furthermore, it provides a sub-

model for reward distribution which is named as fair agent reward distribution based dynamic feature selection FARD-DFS.

3. It provides four reward distribution methods for FARD-DFS, namely, accumulative contribution (ACC), alternative contribution (ALC) and impurity based (IM), and one action at one time (OA-OT). Furthermore, it allows simultaneous action change to the agents, which enables fast exploring of the solution space and increases the efficiency of the learning.
4. It proposes spatial-temporal dynamic feature selection using tabu search algorithm with agglomerative clustering algorithm to enable feature selection based on temporal and spatial information of the instances jointly. The proposed model called Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS). The tabu search algorithm is formulated to select the optimal relevant sub-set features over time, where the agglomerative algorithm employed to measure the similarity distance of samples in the same chunk and distributed them into a set of clusters to select optimal sub-set features spatially at cluster level. 2TS-DFS was applied on hidden state features extracted from dynamic word embedding and the selected features are fed into a TextCNN based classifier.

1.6 Research Scope

The scope of this study is primarily focused on proposing an adaptive approach for Cross-Site Scripting (XSS) attack detection using machine learning (ML) and deep learning (DL) techniques. XSS attacks are a prevalent security threat in web applications, where malicious scripts are injected into benign and trusted websites. These attacks can lead to data theft, session hijacking, and other severe consequences. The criticality and increasing sophistication of XSS attacks necessitate the development of advanced detection mechanisms. It is important to note that this research does not explore other types of cyber-attacks or the real-time implementation of the proposed methods in live environments, which could be areas for future research.

The scope of the study is confined to two types of datasets that are commonly associated with XSS attacks: textual data, including JavaScript and payload datasets, and numerical datasets. These datasets were specifically chosen due to their relevance in typical XSS attack vectors, where attackers can easily embed malicious scripts. Detecting these scripts is particularly challenging, making these datasets ideal for testing the robustness of the proposed methods. The datasets are processed sequentially, meaning that data is handled in chunks over time. This approach simulates the evolution of attacks, allowing for incremental learning and enabling the study to capture both temporal and spatial feature drift accurately because these datasets are a real-world data that consist of a different combination of drifts.

This research focuses on resolving the XSS attacks from the perspective of dynamic feature selection to improve the evolved XSS attack detection in spatial and temporal feature drift. Two core approaches are developed within this research to comprehensively address the diverse XSS data types. One approach is designed for numerical data, while the other focuses on textual data (JavaScript and payloads). While these approaches target different data types, they are not isolated; rather, they function within a unified detection framework and only one approach is activated based on input data type. By integrating these approaches, the study presents a robust framework solution for detecting and adapting to the dynamic and evolving nature of XSS attacks.

For temporal drift using numerical data, we propose a multi-agent reinforcement learning (RL) approach that enables simultaneous feature selection and classification, encapsulated in the Deep Q-network Multi-Agent Feature Selection (DQN-MAFS)

model. This includes innovative reward distribution methods to enhance learning efficiency. For addressing both temporal and spatial feature drift using raw textual data, we introduce a Tabu Search-based Tempo-Spatial Dynamic Feature Selection (2TS-DFS) model, which combines the Tabu search algorithm, agglomerative clustering, and dynamic word embedding methods to optimize feature selection over time and across clusters.

1.7 Thesis Organization

This section describes the organization of the rest of thesis as follows:

Chapter 2: In this chapter, a comprehensive analysis of Cross-Site Scripting (XSS) attack detection is conducted. The chapter covers the background of Cyber Security, Web applications, and Machine learning approaches relevant to XSS attacks. It delves into the different types of XSS attacks, their occurrence scenarios, and the evading strategies associated with them. The classic detection approaches, including static, dynamic, and hybrid analysis, are discussed in detail. Furthermore, the chapter provides an extensive review of machine learning and deep learning approaches employed in previous works for XSS attack detection, encompassing data pre-processing techniques, feature extraction, and feature selection. The concept of concept drift is introduced, with a specific focus on feature drift and its detection methods, particularly reinforcement learning-based dynamic feature selection. The chapter concludes with a discussion of research challenges and limitations identified through the analysis and review, laying the groundwork for the subsequent chapters to propose an approach for XSS attack detection.

Chapter 3: The methodology of the proposed framework is presented in this chapter, covering the problem formulation, an overview of the framework. It provides a discussion of the overview of the general proposed framework.

Chapter 4: In this chapter, the models and sub-models of the framework are explained in detail. It presents the features extraction method and dynamic feature selection methods that are used for temporal and spatial drift detection and employed to differentiate between malicious and benign scripts. The classifiers were used for detection. Additionally, the chapter provides insights into the datasets used, the performance metrics employed for evaluation, and the experimental design implemented. The chapter concludes with a summary of the proposed framework and the methods employed in this thesis, setting the stage for the subsequent chapters' in-depth analysis and discussion of the results.

Chapter 5: Provides the results and discussion of the dynamic and adaptive XSS attack detection framework (DA-XSS-AD) and its sub-models are presented. The chapter is organized into sections and subsections to comprehensively analyze the performance and effectiveness of the framework. It provides the results of the first sub-model, DQN-MAFS, which captures temporal dynamic feature selection and maintains accurate predictions. Furthermore, it delves into the results of the second sub-model, 2TS-DFS, which emphasizes temporal and spatial feature selection and evaluates feature reduction using the tabu search algorithm. In addition, the overall results of the DA-XSS-AD framework, including accuracy, F1-scores, and feature reduction rates across different data types, are discussed. Finally, detailed discussions

and comparisons with existing methods are provided throughout the chapter, culminating in a summary and conclusion.

Chapter 6: Provides a summary of the thesis as a whole, and discusses the main results achieved. In addition, it discusses the results obtained from the experiments and the achievements of this research in relation to its set objectives. Moreover, it summarizes the contribution of this present research and explains the limitations which emerged during this study. Furthermore, it proposes future work related to this thesis.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This research aims at proposing a novel method for dynamic aware XSS detection. In this context, this chapter serves the goal of providing all the necessary background for accomplishing the research objectives and reviewing the existing literature of XSS in order to identify the research gap that aimed by fulfilled through the proposed objectives as depicted in Figure 2.1.

This chapter will carry out an in-depth background and analysis on the Cross-site Scripting (XSS) attacks detection. The chapter begins by introducing the background on the Cyber Security, Web application attacks. Then, it presents insights into XSS attacks, their various types, scenarios and the forms in which they occur. Additionally, the main evading strategies that contribute to the evolution of XSS attacks are identified, as well as the classic XSS detection approaches. Furthermore, the chapter carried out a comprehensive review on the machine learning and deep learning approaches that have been utilized in previous works for leveraging the XSS attacks detection. This review includes different perspectives of data pre-processing approaches, such as data cleaning, feature extraction, and feature selection methods. The current chapter also provides a background on concept drift issues and highlights a recent emerging topic in concept drift, namely feature drift and its various types. In addition, it demonstrates the methods used for detecting the feature drift such as

dynamic feature selection, in more specific, Reinforcement learning-based approaches and applied for dynamic feature selection. Finally, this chapter discusses the research challenges and limitations come up from the analysis and review conducted. The insights gained from this literature survey will serve as a foundation for the subsequent chapter in this thesis, where we propose and develop an approach for XSS attack detection.



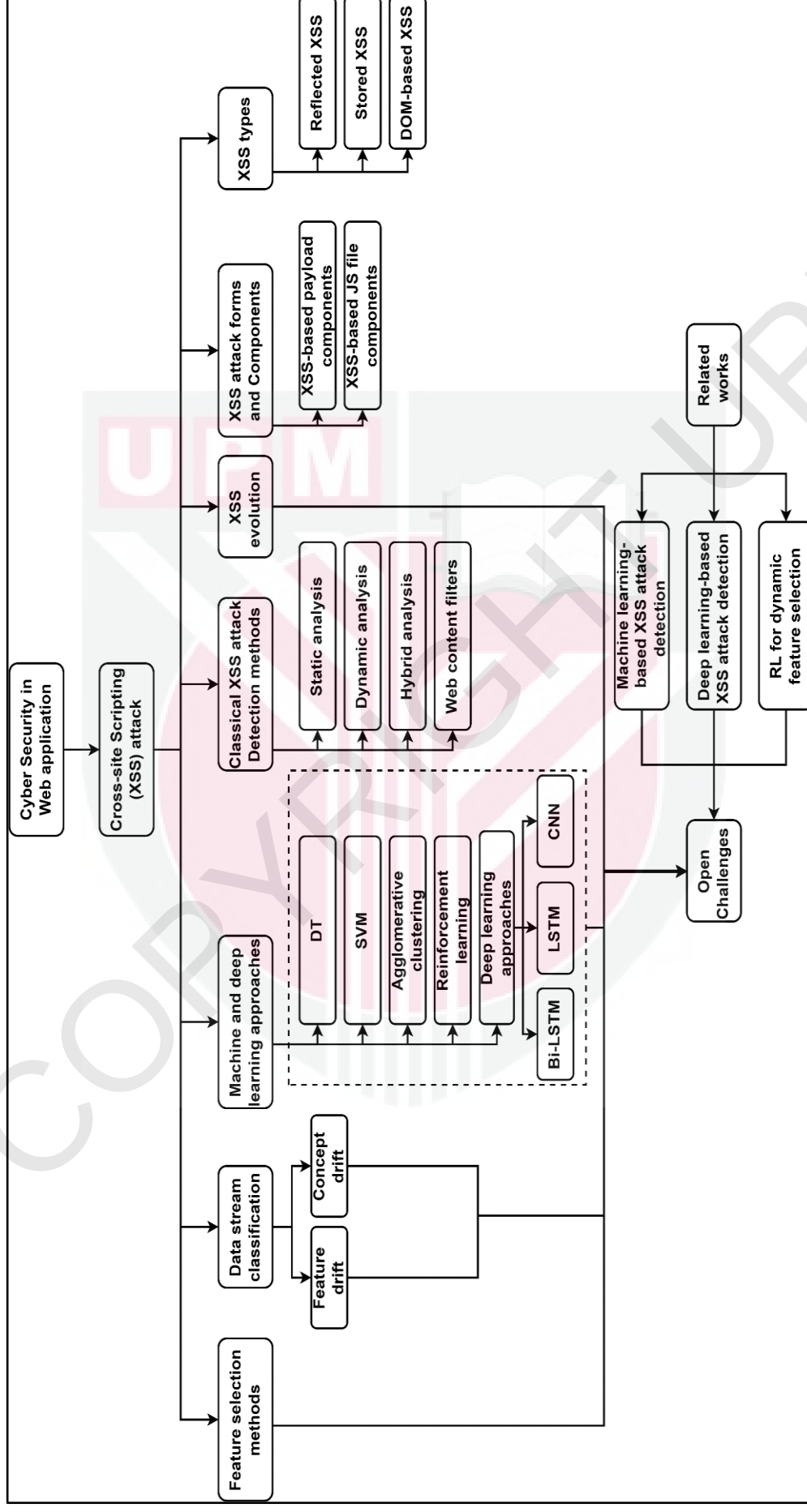


Figure 2.1: Literature Review Map

2.2 Cyber Security in The Web Application Domain

In today's digital era, cyberspace plays an indispensable role in information sharing and various professional activities, encompassing areas like business, e-commerce, banking, advertising, and services. However, the widespread use of the internet has led to a significant rise in cybercriminal activities, with web applications being a common target due to their inherent design flaws (Humayun et al., 2020). Cybercriminals exploit these flaws to gain unauthorized access to applications (Lun et al., 2016). Consequently, cybersecurity has emerged as a paramount concern for both researchers and practitioners. Cybersecurity comprises a multitude of tools, techniques, policies, security measures, guidelines, risk mitigation strategies, actions, training, best practices, and cutting-edge technologies aimed at safeguarding cyberspace and user assets (Von Solms & Van Niekerk, 2013). It involves the protection of digital information through the detection, prevention, and response to cyberattacks (Bada et al., 2019; McAlaney et al., 2018). Within the realm of cybersecurity, web application security assumes particular importance given the pervasive use of web applications as platforms for delivering services to users (Humayun et al., 2020). These applications serve diverse purposes, and often storing sensitive user data like personal information, financial records, and login credentials. However, web applications are susceptible to various cyber threats, and a successful attack can result in severe consequences, including financial losses, damage to reputation, and legal consequences. Vulnerabilities in web applications represent weaknesses or gaps that malicious attackers can exploit, thereby putting application users or developers at risk. As indicated by statistics from the Open Web Application Security Project (OWASP) (OWASP, 2021), the top 10 web application vulnerabilities include:

2021-A1-Broken Access Control: This critical vulnerability enables attackers to bypass access controls, gaining unauthorized entry to restricted pages to steal sensitive data or executing malicious activities. Such vulnerabilities can result from design, implementation, or configuration flaws in web application or API access controls.

2021-A2-Cryptographic Failures: Formerly known as "*Sensitive Data Exposure*," this vulnerability arises from improper use of cryptographic functions, leading to sensitive data exposure. Causes include weak cryptographic protocols, poor key management, and failure to validate server certificates properly.

2021-A3-Injection Attacks: This category encompasses various attacks, including Cross-Site Scripting (XSS), SQL injection, command injection, NoSQL, OS command, Object Relational Mapping (ORM), LDAP, and Expression Language (EL) or Object Graph Navigation Library (OGNL) injections. Among these, XSS attacks are the most common. These vulnerabilities allow attackers to inject malicious scripts or queries into vulnerable applications through untrustworthy data, bypassing proper validation, filtering, sanitization, or context-aware evading.

2021-A4-Insecure Design: A new category highlighting risks related to design and architectural flaws in software development. It underscores the importance of threat modelling, secure design patterns, and reference architectures to ensure secure design principles are met. It is crucial to differentiate design flaws from implementation defects, as they require different remedies. Insecure design often results from failing to perform business risk profiling during software development.

2021-A5-Security Misconfiguration: Arising from system component misconfigurations or outdated components, this vulnerability occurs when security hardening is missing, cloud service permissions are improperly configured, unnecessary features are enabled, default accounts and passwords are not changed, error handling reveals informative messages, or security settings are not appropriately configured.

2021-A6-Vulnerable and Outdated Components: This category focuses on risks related to outdated and vulnerable software components. It includes the use of unmaintained third-party components, posing a risk if organizations are unaware of component versions, use vulnerable or outdated software, lack regular vulnerability scans, and neglect platform and dependency updates.

2021-A7-Identification and Authentication Failures: Emphasizes the importance of verifying user identity and securing authentication processes. Authentication weaknesses may occur if an application allows automated attacks, lacks effective credential recovery, uses plaintext or weakly hashed passwords, misses multi-factor authentication, or has insecure session management practices.

2021-A8-Software and Data Integrity Failures: A new category concentrating on software updates and critical data. Vulnerabilities may result from using untrusted sources, insecure (Continuous Integration (CI)/Continuous Deployment (or Delivery) (CD)) pipelines, or inadequate update verification. Auto-update features lacking sufficient integrity verification can lead to unauthorized accesses or system compromises. Insecure data serialization may allow attackers to view and modify sensitive information.

2021-A9-Security Logging and Monitoring Failures: This category aids in detecting and responding to breaches. Failures occur when logs are not stored or monitored effectively, generate unclear messages, lack appropriate response escalation processes, do not trigger alerts during penetration testing, or dynamic application security testing.

2021-A10-Server-Side Request Forgery (SSRF): SSRF arises when web applications fetch remote resources without proper user-supplied URL validation. Attackers exploit this by forcing the application into sending crafted requests to undesirable destinations. SSRF incidence is increasing due to modern web applications' convenience and complex architectures, heightening the severity of these attacks, particularly in cloud service environments.

Among these vulnerabilities, XSS attacks are both most prevalent and perilous (C. Gupta et al., 2022; Sarmah et al., 2020; Tariq et al., 2021; Y. Zhou & Wang, 2019). XSS attacks involve injecting malicious scripts into web pages through vulnerabilities in web application code, allowing attackers to steal sensitive data, redirect users to phishing sites, and perform other malicious actions. As shown in Figure 2.2. and according to OWASP statistics, XSS attacks have risen in prominence, ranking seventh in 2017 and third in the list of top web application threats in 2021. They pose a constant threat to web application security. In the subsequent section, a detailed description of XSS attacks will be presented.

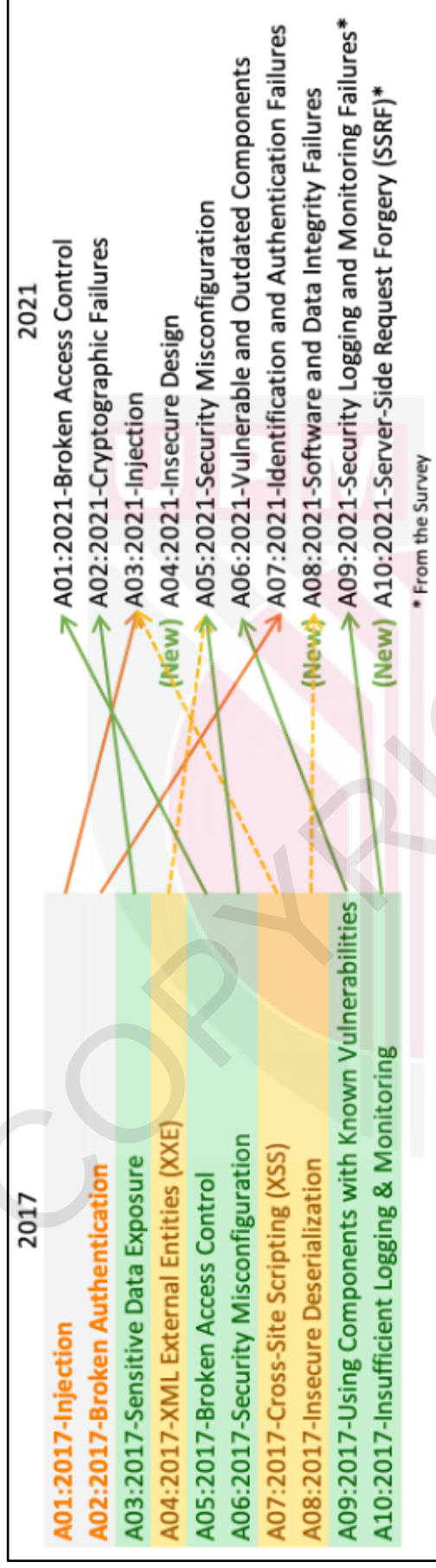


Figure 2.2: OWASP high risk vulnerability statistics and new categories
(Source: OWASP, 2021)

2.3 Cross-site Scripting Attacks (XSS)

Cross-site scripting vulnerability, commonly referred to as XSS, is a type of flaw that poses a significant threat to web applications by injecting malicious code at the application layer. The term "Cross-site scripting" was introduced by Microsoft security engineers in January 2000 (CERT CC, 2001; MSE, 2000) to describe this form vulnerability, distinct from Cascading Style Sheets (CSS). In these attacks, attackers search for vulnerabilities within web applications and exploit them by injecting harmful content into trusted contexts, specifically targeting end users of the application. When legitimate requests are made by users (victims), the affected application serves a page containing the injected malicious code. Unknowingly, the victim's web browser executes this code as if it were a legitimate part of the web application, potentially leading to severe consequences, such as the unauthorized disclosure of sensitive information to the attacker's server (Sarmah et al., 2018). Notably, the success of XSS attacks heavily relies on web applications failing to validate or properly sanitize the users' inputs, particularly when generating output that utilizes raw and unvalidated data are provided by users.

XSS attacks leverage different technologies like ActiveX, HTML, Flash, or JavaScript, with JavaScript being the most commonly exploited one. Approximately 93.6% of all the websites in the worldwide use JavaScript, making it a prime target for attackers (Q-Success, n.d.). Malicious JavaScript snippets exploit the Same Origin Policy, allowing content from a web page to access a system's resources if the origin site possesses the necessary permissions. As a result, the client's browser becomes vulnerable to the attacker's malicious intentions, as it struggles to differentiate between

legitimate and harmful content originating from the same website. In other words, such attacks exploit the failure of developers to properly validate and filter user input or URL submissions, allowing malicious scripts to be injected and executed within the DOM tree, creating a vulnerability. For example, an attacker can inject a payload like "<img%20src=1%20onerror=alert(1a)>" into the parameters of a GET/POST request, causing a pop-up window with the attacker's malicious code to appear on the victim's web page. Thus, through XSS attacks, cybercriminals can compromise the security of the client-side browser, gaining control over passwords, login cookies, authentication tokens, and performing actions on behalf of users, such as sending emails or initiating financial transactions (X. Song et al., 2023). In the next sub-section, we present an overview of the various types of XSS.

2.3.1 XSS Types

XSS vulnerabilities encompass three primary types: Reflected XSS attacks, Stored XSS attacks, and DOM-based XSS attacks. While these types may overlap in certain aspects (Y. Zhou & Wang, 2019), a key distinction among them lies in where untrusted user-supplied data is included, whether it is within an HTTP response generated by the server, an HTTP request generated by the user's browser, or within the DOM of HTML pages (S. Gupta & Gupta, 2017; Hydera et al., 2015; Sarmah et al., 2018). Consequently, XSS attacks can be executed in various ways depending on their type, which these types are briefly explained below:

2.3.1.1 Reflected XSS Attacks

Reflected XSS attacks, also known as Non-persistent XSS attacks or Type-I XSS attacks, involve the attacker cleverly crafting a malicious link that contains a script and then enticing the victim to click on it. Consequently, the victim's request to the server includes the malicious script as a part of it. As a result, when the server responds, it incorporates the malicious code snippet and the victim's browser unknowingly executes it. This type of XSS attack arises when user input is directly included in the application's output without undergoing proper sanitization, rendering the application vulnerable to exploitation. In Figure 2.3, the steps of a Reflected XSS attack are presented.

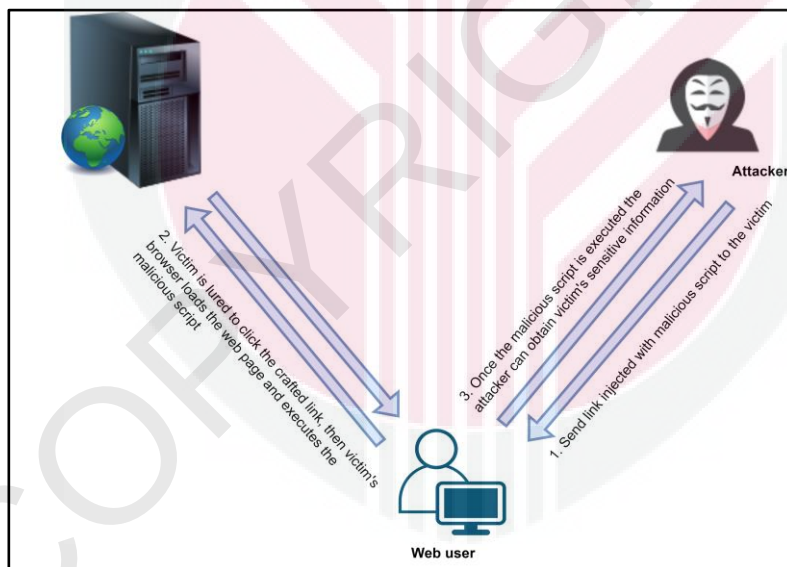


Figure 2.3: Reflected XSS attack steps

2.3.1.2 Stored XSS Attacks

Stored XSS attacks, also referred to as Type-II or Persistent (Continuous) XSS attacks, pose a significant threat to web applications. In this form of attack, the attacker

skillfully designs a malicious code snippet, which is then permanently stored on vulnerable servers. Typically, the injection of this code occurs through message forums or blog posts, after which it becomes part of the application's server. When unsuspecting victims access the compromised site, the malicious code snippet becomes embedded in the web page served to them. Consequently, the victim's browser unknowingly executes the malicious code, exposing them to serious risks. The steps of Stored XSS attack are depicted in Figure 2.4.

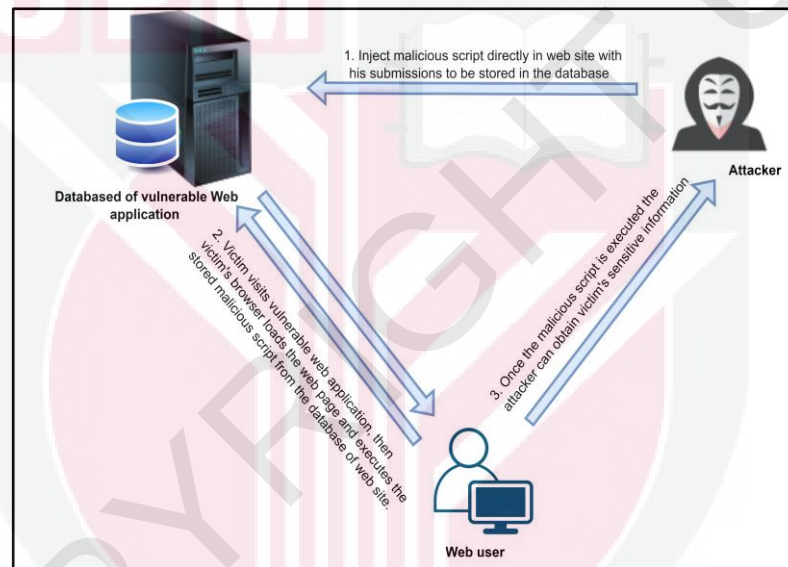


Figure 2.4: Stored XSS attack steps

2.3.1.3 DOM-based XSS Attacks

DOM-Based XSS attacks, also known as Type-0 XSS attacks, originate from client-side vulnerabilities rooted in the DOM. Such vulnerabilities typically occur on web pages containing various DOM objects, which users can manipulate, including URIs, locations, and JavaScript functions like `document.write()` or `eval()`. The DOM allows dynamic modification and examination of page content via client-side scripting. To

execute this attack, the attackers craft a link containing malicious JavaScript code and delivers it to the victim. When the victim allured to click on the link, they receive a harmless response with no evident malicious content. However, the browser can locally execute the acquired data within the DOM. Importantly, unlike reflected XSS, DOM-Based XSS submissions are intended for interactive requests within the browser, bypassing communication with the server. The steps of DOM-based XSS attack are shown in Figure 2.5.

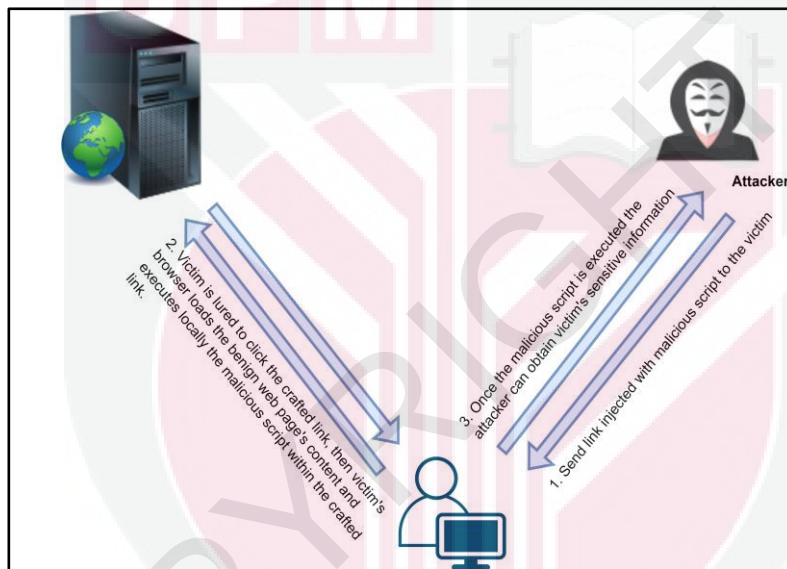


Figure 2.5: DOM-based XSS attack steps

This thesis avoids exclusive focus on one specific type of XSS attack because these attack types often exhibit overlaps and are interconnected (Tariq et al., 2021; Y. Zhou & Wang, 2019). The research encompasses all forms of XSS attacks due to the presence of various XSS types in the datasets used. In the next Section, we present an overview of the various forms and main components of XSS vulnerabilities.

2.3.2 XSS Attack Forms and Components

XSS attacks scenarios are carried out in three forms, each utilizing specific methods to compromise a web application. These forms involve (1) exploiting a maliciously injected URL, (2) injecting a website's content with malicious script, and (3) storing malicious JS file within the web applications' database. The first two attack forms rely on an XSS attack payload vector used to be embedded into the URL or website's contents. This payload encompasses a combination of malicious JS functions, HTML attributes, HTML event handlers embedded within HTML tags. We refer to this type of the attack scenario as Payload-based XSS. Conversely, in the context of the storing a malicious JS file into web application, the XSS attack can be occurred by crafting a malicious JS file is written entirely in JS language, devoid of any of HTML tags. This particular form of attack is known as JS-file-based XSS. In forthcoming subsections, the components of both payload-based XSS and JS-file-based XSS will be elaborated.

2.3.2.1 Payload-based XSS Components

The essential component of Payload-based XSS attacks is the malicious code injected into web applications. Attackers have devised an extensive array of payload attack vectors, employing specific semantic rules and characteristics. In-depth research and analysis conducted by several authors (Y. Fang et al., 2019; Z. Liu et al., 2022; H. Yan et al., 2022; X. Zhang et al., 2020), have revealed that the components and the structure of the attack vector must adhere to the context of the output point to ensure normal parsing by the browser. Typically, the XSS attack vector incorporates HTML tags, attributes, special characters, or events, as depicted in Table 2.1 and examples in Figure 2.6.

Table 2.1: Payload-based XSS components with examples

ID	Components	Examples
1	HTML tags	<a>, <p>, <body>, <script>, <var>, <div>, <object>, <input>, <select>, <iframe>, <frameset>, <embed>, <svg>, <video>, and <audio>.
2	HTML attributes	src, dynsrc, lowsrc, href, action, data, background, formaction, poster, code, location, name.
3	Malicious code	alert(), confirm(), prompt(), self.location, top.location, and location.href., getElementById, createElement
4	Special characters	Termination tags (<\w+>), start tags (<\w+>), window activation (\w+=), functions (?x)[\w\.\!+?\\(
5	Events handlers	onerror, onclick, onblur, onmousedown, onmouseup, onmouseover, and onmousemove.

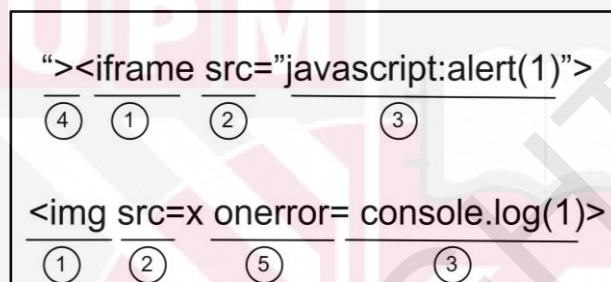


Figure 2.6: Attack vector payload sample with components' labels correspond to Table 2.1

(Source: Z. Liu et al., 2022)

The following is the summarized definitions of the Payload-based XSS components:

- HTML tags:** in HTML, certain attributes can be related to specific HTML tags, making all tags that support such attributes potential elements of XSS attack vectors. Attackers often utilize vulnerable tags to carry out XSS attacks.
- HTML Attributes:** various unique attributes can be employed by attacker to launch the browser's JS parser, examine, and run the JS attack code included in the attack vector, thereby facilitating XSS attacks. Commonly used HTML attributes are listed in Table 2.1.
- Malicious code:** At the heart of the attack vector lies the malicious code, primarily composed of JavaScript. This code varies based on the attacker's intended purpose. It typically includes functions, such as pop-up and jump JavaScript functions, which may serve as essential component of the attack vector, as indicated in Table 2.1. These functions can be embedded within the Script or HTML tags for Payload-based XSS attack or within an external JavaScript file for JS-file-based XSS attack.

- d. **Special characters:** Special characters, particularly termination tag or start tag characters, play a crucial role in the XSS attack vector. Proper selection of these symbols is essential to effectively manipulate the original HTML or DOM structure during the creation of an XSS attack. By ensuring that the chosen characters align with the context of the output location, the attack code within the vector can be executed successfully.
- e. **Events handlers:** HTML tags feature event-driven attributes, where JavaScript code can be executed upon certain events. Some events, like `onerror`, directly launch execution, while others, such as `onmouseover`, require activation after the mouse passes over the specific location in the web page. Table 2.1 presents common events used in XSS attacks.

2.3.2.2 JS Files-based XSS Components

XSS attacks can take advantage of web applications' reliance on external JavaScript files. These files are commonly used to enable various functionalities, such as form validation, dynamic content updates, and user interactions. If attackers manage to inject malicious code into these external JavaScript files, then they inject such files into the web server, the malicious code can be executed whenever the affected file is loaded by a browser (S. Gupta & Gupta, 2016; Sarmah et al., 2018). This scenario can lead to widespread malicious XSS across multiple pages or sections of the website that reference the compromised JavaScript file. As an example of these attacks JavaScript worm attack, including Orkut XSS worm, and Renren worm or W32/PinkRen-A worm (Sarmah et al., 2018). Here are some commonly targeted JavaScript elements that can be exploited for XSS attacks (Ndichu et al., 2019; Park et al., 2016; X. Y. Song et al., 2020):

innerHTML property: If user input is directly assigned to the `innerHTML` property of an HTML element without proper validation and encoding, it can result in an XSS vulnerability by getting the content (JS functions) in the specified HTML tag.

eval() function: The eval() function can execute dynamically a string as a script code.

If user input is directly passed to eval() without proper sanitization, it can lead to code injection and potential XSS attacks.

document.write() method: User input that is directly passed to the document.write() method without proper encoding can write HTML or JavaScript code into an HTML document and result in script injection.

Event handlers: Attacker-controlled code can be injected into event handlers like window.onerror, window.onload or etc. These functions can be called when JS code is loaded or error is occurred. If user input is not properly sanitized before being assigned to these functions, it can lead to XSS attacks.

setAttribute() method: If untrusted data is directly assigned to attributes using the setAttribute() method, it can be exploited for XSS attacks. For example, element.setAttribute('src', userInput) or using function calls setCookie() to access and manipulate the cookie of the user.

location object: Manipulating the location object can lead to various attacks, such as redirecting users to a malicious website (location.replace()) or modifying the URL parameters to execute arbitrary JavaScript.

XMLHttpRequest object: If untrusted data is used to construct AJAX requests without proper validation, an attacker may be able to inject malicious scripts through response manipulation.

encodeURIComponent() function: If user input is not properly encoded using encodeURIComponent() before being included in dynamically generated URLs, it can lead to XSS vulnerabilities.

DOM manipulation methods: Functions like createElement(), appendChild(), getElementById or insertAdjacentHTML() can be used to inject malicious code and manipulate HTML documents by changing the DOM tree if untrusted data is directly passed without proper encoding.

location.href and window.open() methods: If untrusted data is used to construct URLs for redirection or opening new browser windows, it can be exploited for XSS attacks.

We presented the components of these two forms of XSS attack to comprehensively understand the diverse ways in which XSS attacks can occur and to strengthen the capacity of our proposed method. By examining the specific HTML tags, attributes, events, and special characters used in payload-based XSS, and understanding the JavaScript functions, methods, and elements targeted in JS-file-based XSS, we aim to enhance the effectiveness of our detection approach. Leveraging two types of data allows us to address the different scenarios of XSS attacks and develop a more robust defense against this prevalent security threat in web applications.

2.3.3 XSS Evolution

In the ever-evolving landscape of web applications, cybercriminals continuously develop and adapt their tactics to bypass detection systems and exploit vulnerabilities. As security systems detect new attacks, attackers swiftly evolve their techniques,

resulting in a new breed of XSS attacks - a fusion of innovative and aggressive methods that blend various features/components from both benign and malicious XSS records (Gupta et al., 2022). Extensive research into various web filtering mechanisms and attack vector mutation methods has revealed four commonly used mutation techniques that facilitate the evolution of existing XSS attacks into more sophisticated forms (Kuppa et al., 2022; Z. Liu et al., 2022; Tariq et al., 2021). In the following, we delve into each of these four mutation methods in detail to shed light on their mechanisms and implications for the security landscape:

1. Coding Confusion:

Table 2.2: After and before applying the encoding methods

2. Sensitive Words Substitution:

This mutation method involves various techniques used by attackers to replace or swap words and components in XSS attack vector, effectively evading detection and bypassing security measures. The following are the specific techniques employed in this method: substitution of sensitive events, substitution of sensitive JS functions, substituting blank characters, and substitution of brackets.

3. Transforming Position and Form:

Attackers employ various techniques to alter the position and structure of XSS components, ingeniously evading detection mechanisms and circumventing security measures. The following specific techniques are utilized in this method: swapping attributes and events, swapping the letter-case (upper-case or lower-case), and transforming the shape of pop-up window functions (splitting or add more special characters).

4. Insertion of Special Characters:

Attackers utilize a diverse array of techniques of inserting blank characters, nested tags, adding comments, or including random characters within their XSS attack vectors, skillfully evading detection mechanisms. The following specific methods exemplify how this approach is implemented: blank character insertion, nested tag insertion, comment addition, and random characters insertion.

The presented XSS evolution techniques are essential for addressing the persistent challenge posed by attackers who continually find ways to bypass security measures

and launch evolved XSS attacks. Despite the efforts of developers and detection systems to enhance web application security through content filtering and XSS prevention, attackers adapt and employ mutation methods to evade detection and cause feature drift. This feature drift indicates a change in the distribution representing the source of data, leading to changes in the relevancy of XSS attack features over time. Evolved XSS attacks are characterized as confusing and aggressive, employing innovative combinations of features from existing benign and malicious XSS records. The primary objective of this thesis is to develop a robust and efficient method for detecting and mitigating evolving XSS attacks, leveraging an understanding of the evolving nature of these attacks and the mutation techniques employed by attackers. In subsequent section the classical methods employed for XSS attack detection will be presented.

2.4 Classical XSS Attack Detection Methods

In this section, we present an overview of the classical techniques have been utilized in the literature for analysing and detecting XSS attacks, assessing the merits and limitations of each approach. The analysis of XSS attacks is primarily focused on the analysis paradigm employed, including static, dynamic, hybrid analysis, web application firewall, and content filtering (Melicher et al., 2018; Melicher & Fung, 2021). By exploring these initial methods and their specific characteristics, we gain insights into the XSS attack detection and the challenges they sought to address.

2.4.1 Static Analysis

Static analysis, also known as white-box testing, is a widely used approach in the context of XSS vulnerability detection. It involves utilizing static tools for source code analysis without running the code. Techniques like taint analysis and data flow analysis are employed to deduce attributes of the source code or object (Usha et al., 2020). For detecting malicious JavaScript instances, static analysis is preferred due to its speed, reliability, and code coverage. However, it has a drawback of high false negative rates (He et al., 2018). Static Taint Analysis, a method within static analysis, utilizes data flow analysis to track information transfer from user-controllable inputs (sources) to various code processing or display points (sinks) (Kronjee et al., 2018; Staicu et al., 2019). Its goal is to identify potential vulnerabilities where malicious scripts can be injected through user inputs and propagate to sensitive parts of the application. String Analysis is another technique used in static analysis to scrutinize source code strings in HTML or JavaScript for embedded errors (Christensen et al., 2003). It involves feature (character or keyword) extraction from the code using semantic and syntax-based methods and transforms the code into representations for analysis (X. Y. Song et al., 2020). Additionally, Static Software Testing Techniques are employed as preventive measures during the development cycle to identify software issues before code deployment, with various commercial tools available for JavaScript, each offering unique findings based on their specific approaches (M. Mohammadi et al., 2017, 2019). While these methods are valuable, they may not alone be sufficient to detect all XSS vulnerabilities, as web applications often interpret their internal scripts using a JavaScript interpreter or HTML parser. Also, it cost resources when additional necessary runtime filters are incorporated to ensure protection against all possible injection vectors.

2.4.2 Dynamic Analysis

Dynamic Analysis, often referred to as code execution and behavior monitoring, stands in contrast to static analysis. It focuses on capturing the runtime behavior of an application's code, offering a deeper understanding of its actual operations. This method aims to detect and identify malicious code or malware during real-time code execution, providing real-time threat detection (Stock et al., 2014). Dynamic analysis entails modifying browser interpreters to monitor code execution behavior. Notable studies include one using an instrumenting interpreter for character-level data processing monitoring, offering accurate vulnerability detection (Pietraszek & Berghe, 2006). Another introduced the TT-XSS framework for DOM-Based XSS detection, consisting of modules for URL collection, taint tracking analysis, and vulnerability verification (R. Wang et al., 2018a). Additionally, a dynamic taint flow tracking technique using the Chromium browser and V8 JavaScript engine collected extensive JavaScript functions for vulnerability prediction (Melicher et al., 2018; Melicher & Fung, 2021). While dynamic analysis is valuable for detecting XSS attack, it requires more time, resources, and expertise to develop interpreter engines, which can be challenging for certain programming languages like Java. In some cases, dynamic analysis may not produce results within an acceptable timeframe (M. Liu et al., 2019; X. Y. Song et al., 2020).

2.4.3 Hybrid Analysis

Hybrid analysis is a proactive strategy aimed at mitigating the drawbacks of static analysis, particularly false negatives. Researchers have explored integration of static

and dynamic analysis techniques to enhance XSS detection. For instance, Hybrid XSS detection (HxD) tool, introduced by (Choi et al., 2017), leverages PhantomJS, a headless browser, to combine both approaches. HxD employs a comprehensive framework consisting of a Log Analyzer, Data Manager, XSS Detector, and Database. The Log Analyzer processes and refines URLs, the Data Manager oversees system operations, and the XSS Detector combines static analysis by injecting XSS payloads into URLs and dynamic analysis using PhantomJS to verify real-time XSS vulnerabilities. Similarly, (Lv et al., 2019) designed a prototype tool utilizing a PhantomJS-based headless browser to assess injection success based on program behavior. However, despite its promise, the hybrid approach inherits the false positive issue from static analysis, making it less favored due to the added complexity in developing browser interpreters alongside the overhead it introduces (Damodaran et al., 2017; Pandey & Alsolami, 2020).

2.4.4 Web Content Filters

The fourth classical method for detecting XSS attacks involves employing web content filters, including content security policies (CSP), web application firewalls (WAF), and web browser filters. Content security policies (Stamm et al., 2010) were used by web developers to restrict script execution on websites, but their effectiveness can be compromised due to misconfigurations or inadequate restrictions on XSS exploits (Calzavara et al., 2021). Another experimental browser API called trusted types, introduced in 2019, aims to enhance XSS protection by requiring JavaScript code to access sensitive sinks through a special CSP header. However, its implementation requires code modifications and may not be compatible with legacy code (Kotowicz,

2020). Web application firewalls (WAF) are commonly used for defense against web vulnerabilities by scrutinizing network-level requests for potential exploits. However, attackers can sometimes evade these filters by altering their exploits to bypass detection patterns using encodings like HTML or XML encoding (Melicher et al., 2021; Vladimir Ivanov, 2016). Client-side XSS filters in modern web browsers, such as IE XSS filter (Microsoft, 2008), XSS Auditor (Chromium, 2019), and NoScript (Giorgio Maone, 2005), aim to block reflected XSS attacks but have limitations against stored XSS attacks (S. Gupta & Gupta, 2019). These filters rely on string matching with predefined rules to detect XSS code strings but can be bypassed through various techniques. Overall, web content filters provide a defense layer against XSS, but they may have limitations and require proper configuration to be effective (Choi et al., 2017; Melicher et al., 2021; Stock et al., 2014). In the subsequent section, an overview of ML techniques used in this thesis, is provided.

2.5 Data Stream Classification

Data stream classification is a crucial component of data stream mining, which involves the real-time or near real-time analysis of continuously generated and vast data volumes. Recent studies project that the number of smart devices connected to web application services will reach nearly 75 billion by 2025, generating approximately 79.4 zettabytes of data (Chaudhary et al., 2021). Data stream mining is essential for extracting valuable insights from this data torrent, particularly due to the resource constraints in processing time and memory space (Sato et al., 2022). Classification is the primary technique for knowledge extraction from data streams (Barddal et al., 2017). In essence, data stream classification is a critical approach for

making sense of ever-growing, sequential data streams across various domains like finance, healthcare, and cybersecurity, facilitating timely and efficient knowledge extraction. The process of classification can be defined as follows: Given a set of n training instances represented by pairs $(\vec{x}_i, y_i)$, where y_i is a discrete class label and assigned for a set of instances based on their relation and affinities, and $\vec{x}_i$ is a d -dimensional vector of attributes belonging to a feature set $\mathcal{D}$ with a cardinality of d , features can be categorical, ordinal, and numeric features. A classifier utilizes the training set to create a model $f; \vec{x} \rightarrow Y$, which can then be applied to classify new, unlabelled instances $\vec{x}_i$ efficiently and accurately based on knowledge of trained model (Bifet et al., 2009). This automated decision-making process aids in predicting class labels for incoming data points based on patterns learned from the training data. Classification can be formulated using Bayesian theory, where decisions rely on prior class probabilities $P[y]$ and conditional probability density functions (*pdfs*) $P[\vec{x}|y]$ for all possible classes $y_i \in Y$ (Gama et al., 2014). In this context, the posterior probability for any arbitrary class y_i and $P[\vec{x}] = \sum_{y_i \in Y} P[y_i] \cdot P[\vec{x}|y_i]$ is given by the Equation (2.1).

$$P[y_i | \vec{x}] = \frac{P[y_i] \cdot P[\vec{x}|y_i]}{P[\vec{x}]} \quad (2.1)$$

2.5.1 Concept Drift

Concept drift is the phenomenon where the statistical properties (distribution underlying) of a data stream change over time, causing the underlying concept or pattern that a machine learning model is intended to learn to shift (Lu et al., 2019). Concept drift poses a significant challenge for machine learning algorithms that are

trained using batch learning techniques. Such techniques assume that there is a static dataset not changed overtime should processed and discarded after training according to limit memory space. Thus, these techniques not able to adapt to changes in the data distribution (Gama et al., 2014; Sato et al., 2022). This can cause the model's performance to degrade, as the assumptions made during training are no longer valid for the new data.

A concept CT is described by Equation (2:2) as a collection of class-conditional probability density functions and recent probabilities for the classes (Barddal et al., 2017; Lu et al., 2019; H. L. Nguyen et al., 2012).

$$CT(s) = \cup_{y_s \in Y} \{P[y_s], P[\vec{x}|y_s]\} \quad (2.2)$$

Where s_t is retrieved samples generated by concept CT_t . In instant situation, the concept is stable $CT_t(s) = CT_{t-1}(s)$. On the other side, concept drift occurs if the concept is shifted over two times-stamps t , and $t + \Delta$, where $\Delta \geq 1$, then $CT_t(s) \neq CT_{t+\Delta}(s)$.

2.5.2 Feature drift

Feature drift is defined as the one of the concept drift forms that caused due to the change of relevance of a feature over time (Barddal et al., 2017). It comes into play when a feature selection algorithm is used on a data arrive continuously, where the non-stationarity of the phenomenon produces a variable subset of best features for classification as times passes. As a result, a learner is forced to modify its prediction

model in order to overlook no longer relevant features and take those who are most relevant into consideration.

$R_t(F): X \rightarrow Y$ is relevancy function that is related to time t where:

F represents the set of features that we want to check their relevancy.

X denotes the candidate combination of features.

Y denotes a decision variable that has value of 0 or 1 where 0 indicates to non-relevancy and one indicates to relevancy relationship.

The feature drift happens if and only if:

$$\exists t_1, t_2 \in [0, t], F \in X, t_1 \neq t_2, \text{ where } R_{t_1}(F) \neq R_{t_2}(F)$$

When a feature drift happens at some moment in the interval $[0, t]$, there are four main types of feature drift (Ferone & Maratea, 2021) . We present them in Figure 2.7.

Sudden drift: It occurs when a specific feature throughout a specific number of time periods drifts from being relevant to irrelevant for classification and vice versa. The feature should either be removed or added to the optimal subset of features based on their relevancy changes. The feature relevancy changes remain in that state for a certain number of consecutive time periods.

Gradual drift: It happens when several successive time periods are considered, and the frequency of a feature being considered relevant or irrelevant changes with time. In this case, a feature initially sporadically emerges and then vanishes from the subset of relevant features before stabilising.

Incremental drift: It happens when the feature relevance within several successive time periods is represented by a serial number in the range of $[0, 1]$ that grows or declines incrementally.

Reoccurring drift: It occurs when a feature loses its relevance suddenly during a specific period and becomes significant again in subsequent periods, and vice versa, with no discernible pattern. If the changeover is rapid, it can turn into a reoccurring shift.

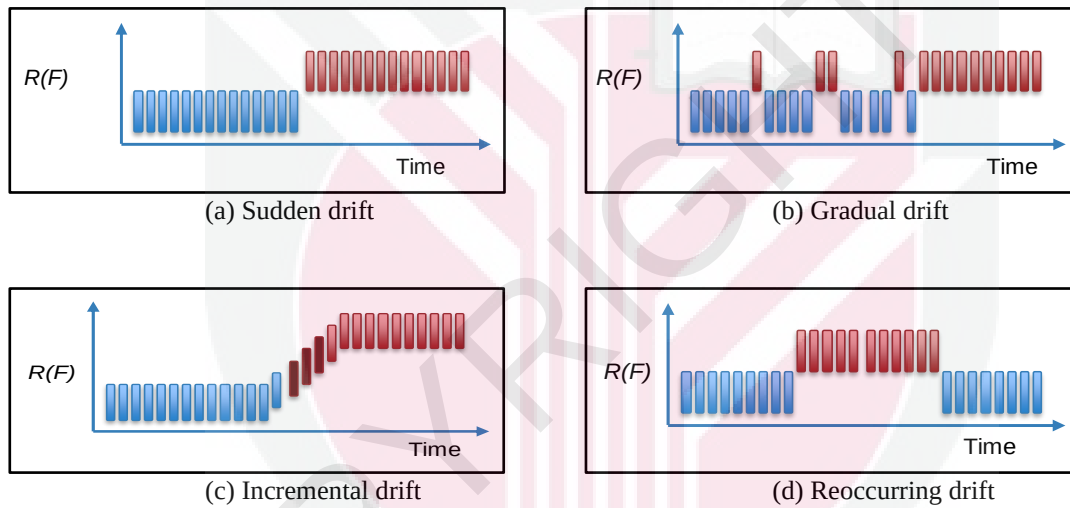


Figure 2.7: Main types of feature drift
 (Source: Ferone & Maratea, 2021)

In a web application environment, data arrives as a continuous stream, and it is processed sequentially based on user interactions. This dynamic nature has transformed web applications into non-stationary environments. Consequently, the vulnerability of web application attack detection systems to feature drift issues has become a concern. This applies to ML/DL-based XSS attack detection systems as well. Feature drift can occur in XSS attacks due to various factors (adopting evasion strategies) as illustrated in Section (2.3.3). Consequently, the relevance of features for

both XSS attacks and benign content can shift over time as attackers adapt their strategies to evade detection. Evolving XSS attacks can exhibit various types of feature drift or even a combination of them. For instance, sudden drifts may occur with new variant attacks, while gradual, incremental, or recurring feature drift can manifest when attackers employ evasion techniques, generating new combination attacks (C. Gupta et al., 2022; Singh, 2020). These novel forms of XSS attacks can be considered sudden drifts led to bypassing server-side and client-side XSS filters (Heiderich et al., 2013).

In the subsequent section, we will provide a comprehensive overview of the feature selection methods utilized for XSS attack detection.

2.6 Feature Selection Methods

Feature selection is the step of choosing most relevant subset features from the entire feature space of the original dataset for using them as input into the ML models. The feature selection process aims to decrease the dimensionality, computational cost, training time of the prediction model, extreme information loss and overfitting (Chandrashekar & Sahin, 2014). Common features can be classified into three types are relevant, irrelevant, and redundant features. Relevant features are statistically correlated with the class label and significantly impact the prediction ability. On the other hand, irrelevant features do not contribute to class prediction, while redundant features are correlated with other features and do not enhance prediction accuracy (Che et al., 2017). Mathematically, we represent this process by considering a selection set of features S_i from the original features $\mathcal{D}$ excluding D_i targeted features. If there exists another subset S'_i :

$\exists S'_i \subset S_i$, such that $P[Y|D_i, S'_i] \neq P[Y|S'_i]$, then is considered relevant, Otherwise, D_i is said to be irrelevant.

Or $P[Y|D_i, S'_i] = P[Y|S_i] \wedge P[Y|S_i] \neq P[Y|S'_i]$, then D_i is considered redundant.

There are various methods for feature selection, such as filters, wrappers, and embedded methods. In the context of XSS attack detection, we have categorized these techniques into five main categories based on our literature review, as follows:

2.6.1 Selection-based Knowledge Methods

In XSS detection, we found that most studies like (Alazab et al., 2022; Cui et al., 2018; X. Li et al., 2020; Niu & Li, 2020; Tariq et al., 2021) rely on their knowledge (domain expertise) in extracting and selecting features. This can be seen clearly in many works where only a certain number of features are extracted and selected in the study without using any formal feature selection methods. Such method has potential limitations such as biased feature selection (may miss the most relevant feature of XSS attacks), limited feature set that does not capture all of attack characteristics, lack of generalizability that result in features are specific to particular dataset or data type, and selecting high feature size can be time-consuming.

2.6.2 Filtering Methods

Such methods evaluate each feature independently and assign a score or ranking based on some criteria such as correlation coefficient, information Gain, Chi-square, or

Document frequency. The correlation coefficient measures the strength and direction of the linear relationship between two variables, and can be used to filter out features that have a weak correlation with the target variable. Information Gain (IG) measures the reduction in entropy of the target variable when a feature is added to the model. Chi-square test measures the dependence between two categorical variables. Document frequency measures the frequency of a term in a collection of documents, and can be used to filter out features that are too rare or too common. In the context of XSS attack detection, study of (Alazab et al., 2022) used correlation coefficient, Information Gain (IG) for selecting relevant features. (Ghaleb et al., 2022) employed Information Gain (uses entropy to measure impurity). (R. Yan et al., 2018) used information gain, Chi-square test (CHI) and document frequency (DF). However, Filtering methods can miss out on potentially important interactions between features, as they evaluate each feature independently (Pham & Raahemi, 2023). Additionally, these methods may not work well for datasets with a large number of features.

2.6.3 Wrapper Methods

These methods evaluate subsets of features by evaluating their combinations using a predictive model with each subset to determine the most relevant features. Though this approach is computationally expensive due to exploring a large feature space multiple times, it considers feature interactions and typically provide more effective feature selection. An example of these methods including Forward Feature Selection (FFS), Backward Feature Elimination (BFE), and Recursive Feature Elimination (RFE). The study of (Mokbal et al., 2021) uses Sequential backward selection (SBS) with IG to select relevant statistical features of XSS attacks. However, a significant issue of such

methods is that the search space for n features is 2^n , which is impractical when number of features n is large (J. Li et al., 2017).

2.6.4 Embedded Methods

Some studies use embedded, incorporating feature selection during model training rather than separate preprocessing step. An example of these methods including Ridge Regression, Lasso Regression, and Random Forest. In domain of XSS attack detection, one study of (Panigrahi et al., 2022) uses Multi-Objective Evolutionary Feature Selection (MOEFS), which is an embedded method. Another study conducted by (X. Li et al., 2020) used the CfsSubsetEval feature evaluation method and a greedy search. Another study of (Tariq et al., 2021) uses a Reinforcement learning algorithm to select/modify the relevant pattern based on the classification error rate with statistical features and domain knowledge. However, Embedded methods can be sensitive to the choice of model and hyperparameters. Additionally, they may not always produce interpretable feature selection results (Pham & Raahemi, 2023).

2.6.5 Attention Models

Attention mechanisms are employed in some studies as a feature selection method, focusing on the most relevant input parts by assigning weights to each feature. They are often used with NLP techniques like word2vec, LSTM, and Bi-LSTM. Attention models help address long-term dependency issues in sequential data, such as language. Such a method was implemented for XSS attack detection in the studies of (Y. Fang, Xu, et al., 2020; Hu et al., 2023; Z. Liu et al., 2023b; Maurel et al., 2022; J. Yang et al., 2020).

In the context of feature selection, attention models have demonstrated considerable success in NLP domains due to their ability to capture complex dependencies between words in a sequence, accounting for the hierarchical and context-dependent nature of language. The effectiveness of attention mechanisms in NLP is largely attributed to their ability to weigh the importance of different tokens based on their context, which is crucial for understanding language semantics. However, attention models may not be as effective for feature selection in non-NLP domains. Non-NLP domains often deal with structured or semi-structured data where the relationships between features are not inherently sequential or context-dependent as they are in language. For instance, in tabular data, features may be independent or have relationships that are better captured by methods that consider global rather than local dependencies. Furthermore, feature selection in non-NLP domains often requires identifying a subset of features that contribute the most to the predictive power of a model. This task is typically more aligned with methods that focus on statistical correlations, mutual information, or other domain-specific heuristics (Yasuda et al., 2022). Attention models, which are primarily designed for tasks like translation, text generation, or sequence labeling, may not naturally align with the goal of identifying and ranking features based on their relevance to the target variable in non-NLP domains.

Overall, the landscape of feature selection methods in XSS attack detection reveals a range of approaches, each with its own set of strengths and limitations. Based on the previous Sections (2.6.1-2.6.5), methods such as: 1- Selection-based knowledge (Alazab et al., 2022; Cui et al., 2018; X. Li et al., 2020; Niu & Li, 2020; Tariq et al., 2021), 2- filtering (Alazab et al., 2022; Ghaleb et al., 2022; R. Yan et al., 2018), 3- wrapper (Mokbal et al., 2021), 4- embedded (X. Li et al., 2020; Panigrahi et al., 2022;

Tariq et al., 2021), and 5- attentions (Y. Fang, Xu, et al., 2020; Hu et al., 2023; Z. Liu et al., 2023b; Maurel et al., 2022; J. Yang et al., 2020). However, a closer examination of these methods uncovers significant gaps and limitations, particularly in addressing the dynamic and evolving nature of XSS attacks, which is critical to the effectiveness of any detection mechanism. While these methods have demonstrated effectiveness in scenarios where data characteristics remain relatively constant, they falter when confronted with the reality of modern XSS attacks. Attackers frequently update their strategies, introducing new variants that exploit different aspects of web security. This evolutionary aspect of XSS attacks means that the relevance of specific features can shift over time, rendering static feature selection approaches insufficient.

Furthermore, these methods are typically based on batch learning, which involves offline training followed by online testing (Barddal et al., 2019; Fahy & Yang, 2019; X. Wang et al., 2022). In such scenarios, feature selection occurs in a static manner, where a fixed number of features is selected for the entire dataset during the offline training phase. This approach is problematic in dynamic environments like XSS detection, where data evolves continuously, and new variants of attacks emerge over time. By selecting features in a fixed manner, these methods fail to account for changes in feature relevance that occur as new data becomes available, limiting their effectiveness to adapt to feature drift—the phenomenon where the importance of features changes over time due to evolving attack patterns. This leads to a critical gap in detection capabilities, as these methods are not equipped to handle the shifting feature landscape that characterizes real-world XSS attacks.

To address these critical shortcomings, this thesis proposes a hybrid feature selection approach that integrates the strengths of existing methods while introducing dynamic adaptability. This hybrid approach not only incorporates the strengths of both wrapper and embedded methods but also addresses the deep-seated issues of static feature selection in dynamic environments. Unlike traditional batch learning methods, which select a fixed set of features based on the entire dataset during offline training, the proposed models incorporate incremental learning. This approach acknowledges that in a real-world setting, the entire dataset is not always available upfront. Instead, data arrives sequentially over time in chunks, each consisting of samples collected within the same time interval. In this proposed design, feature selection is conducted not only at the chunk level (temporal aspect) but also at the cluster level (spatial aspect), allowing the models to dynamically adjust to both temporal and spatial feature drifts. The DQN-MAFS and 2TS-DFS models select relevant subsets of features for each streaming data chunk, integrating the feature selection process into the model training phase. This method evaluates and updates the selected features continuously as new data arrives, ensuring that the detection mechanism remains effective even as the nature of XSS attacks evolves.

2.7 Related Works

In this section, we conduct a comprehensive literature survey focused on studies that are directly pertinent to the core themes of this thesis: Machine Learning (ML) and Deep Learning (DL)-based XSS attack detection, as well as dynamic feature selection (DFS) methods. The selected studies have been chosen based on their relevance to the key challenges and objectives addressed in this thesis, including the detection of evolving XSS attacks and the need for adaptive, dynamic feature selection.

Each study analyzed in this survey offers insights into different methodologies, such as how ML and DL techniques have been applied to XSS detection, and how DFS methods have evolved to handle feature drift in dynamic environments. These studies were carefully selected because they employ similar techniques and address problems closely aligned with those explored in this thesis, providing a foundation for comparison and critical analysis.

By examining these works, this survey aims to not only highlight the strengths and weaknesses of existing approaches but also to clearly delineate the gaps in the literature that this thesis seeks to address. The insights gained from these studies are instrumental in justifying the development and implementation of the novel methodologies proposed in this thesis, and in demonstrating how they advance the state of the art in XSS attack detection and dynamic feature selection.

2.7.1 ML/DL-based XSS Attack Detection Approaches

The literature presents a broad of ML/DL-based methods proposed for the detection of XSS attacks. This section provides a comprehensive review of the machine learning and deep learning techniques employed in XSS attack detection. It is structured into two sub-sections. The first sub-section offers insights into machine learning-based methods applied to XSS attack detection. The second sub-section surveys deep learning-based approaches within the same domain. All reviewed studies will be analyzed in terms of preprocessing, feature extraction, feature selection, and learning models.

2.7.1.1 Machine Learning-based XSS Attack Detection Methods

In the realm of XSS attack detection, several ML approaches have been employed. These approaches encompass a variety of traditional classifiers such as Artificial Neural Network (ANN), Decision Tree (DT), Multilayer Perceptron (MLP), Extreme Gradient Boost (XGBoost), Random Forest (RF), Support Vector Machine (SVM), ensemble learning techniques, and Reinforcement Learning (RL) as shown in the Table 2.3.

Mokbal et al., 2019 proposed a statistical-based feature extraction method coupled with a an MLP scheme for the XSS attack detection. Their feature extraction method employed statistical method using counting mechanism to extract 41 hand-crafted features, including payload characters length, the length of URL, HTML tags apperance, HTML events, HTML attributes, special characters, script tags, URL cookies, JS function, and more. These features were subsequently input into the MLP model for classification.

DT have also found be used in the approach proposed by (Panigrahi et al., 2022) for XSS attack detection. This method is a hybrid model combining of DT and Naive Bayes was used to detect various intrusions, including XSS attacks. They employed Multi-Objective Evolutionary Feature Selection (MOEFS) as an embedded feature selection technique. Ultimately selecting Only 5 features from a total of 83 features.

Random Forest (RF) is another algorithm that researchers have utilized for XSS attack detection. Lu et al., 2022 introduced a fusion verification method that combined traffic detection with XSS payload detection using the RF classifier. Their approach involved

preprocessing data by decoding and converting it to lowercase. Statistical-based feature extraction methods were then applied, resulting in 37 handcrafted features (these features extracted based on the domain-expertise-based method). Another work suggested by V. K. Malviya et al., 2021 different ML classifiers have been adopted with a web browser tool to detect and mitigate XSS attacks. This method extracted 43 features from web pages and JavaScript scripts, with an emphasis on script code features. Various ML classifiers, including SVM, NB, RF, and ADTree, were deployed to classify web pages as malicious or non-malicious. The RF classifier, in particular, achieved impressive performance with 99.5% accuracy, 99.82% precision, 99.74% recall, and a 99.8% F1-score. Similarly, H.-C. Chen et al., 2021 implemented three approaches for real-time predicting of XSS attacks. Their results revealed that the RF algorithm outperformed SVM, K-NN, and LR algorithms, achieving high accuracy, recall, and precision values. However, these methods do not account for incremental learning or feature drift detection in the context of XSS attack detection.

In other several studies, the XGBoost algorithm has been utilized as the base classifier for XSS attack detection. For instance, Rozi et al., 2022 employed XGBoost as main classifier in their model for XSS detection. To extract features, they used the graph-based approach (Grap2Vector), which analyzed the structural features of an AST graph representing the JS program code. This approach allowed them to detect malicious signatures, particularly those characterized by obfuscation techniques implemented with nonstandard tools. The approach presented by Mokbal et al., 2020, 2021 also delved into Extreme Gradient Boost (XGBoost)-based XSS detection. They further refined their approach by optimizing extreme parameters through the Grid-search technique. Their feature extraction method, initially developed in previous works

(Mokbal et al., 2019; Mokbal et al., 2020) was expanded upon in a recent study (Mokbal et al., 2021) to extract a total of 167 features. In this study, an additional component was introduced, combining information gain (IG) with sequential backward selection (SBS) to select the 60 most relevant subset features out of the 167. Their dataset is used as a numerical data for evaluating our proposed DQN-MAFS model. Despite the sophistication of their hybrid feature selection method, it is based on batch learning, where a fixed set of features is selected for the entire dataset.

The SVM algorithm has been widely adopted for distinguishing between normal and malicious attack samples, as demonstrated in studies by (Alazab et al., 2022; Ngoc & Mimura, 2021; Phung & Mimura, 2021). Alazab et al., 2022 proposed an automatic IDS based on the SVM algorithm for JavaScript attack detection. Their features were extracted using a domain-expertise-based method, resulting in 170 extracted features. Correlation coefficient and Information Gain (IG) were then used as filter feature selection methods, selecting 60 relevant features from the original 170 for input into the SVM model. In the method suggested by Phung & Mimura, 2021, an SVM model served as a classifier for malicious JavaScript attack detection. They adopted an NLP-based feature extraction method utilizing the Doc2Vec model to convert raw source code into embedding vector representations and extract features from the textual data. In the same context, Z. Cheng et al., 2021 aimed to compare the performance of two models for XSS attack detection, one employing conventional feature extraction and the other trained within the semantic structure. The data was pre-processed through tokenization to segment payloads into individual components. Four machine learning algorithms, namely RT, DT, SVM, and K-neighbors, were used in the experiment. Results showed a significant enhancement in model performance, particularly with the

SVM model, which demonstrated a 13% improvement in F1-score when utilizing the semantic structure. Additionally, the approach demonstrated in (V. Malviya et al., 2018) introduced a plugin-based feature extraction framework for semi-structured and unstructured data from web pages, scripts, and emails. They performed classification experiments using datasets featuring script-based and web page-based features. Statistical-based feature extraction using a counting approach yielded 34 JS and web page features. Classifiers including RF, DT, SVM, and KNN were employed for classification, with the SVM classifier achieving the best performance in both datasets, boasting F1-scores of 98.1% and 95.4%, respectively. However, SVM's performance tends to degrade with larger datasets due to increased training times, and it struggles with noisy datasets (Ray, 2019). Moreover, SVM's utility in providing probability estimations is limited, making it challenging to interpret the final SVM model.

Another some researchers have used ensemble learning-based approaches for XSS attack detection. Ghaleb et al., 2022 utilized boosting ensemble learning, incorporating three RF, and MLP for final decision-making process. They started by cleaning and transforming the data from its original form to token representation through normalization and tokenization. Features were extracted at both the word and character levels using the N-gram model and the statistical-based text representation, Term Frequency-Inverse Document Frequency (TF-IDF) (Tahmasebi & Risse, 2011). Different datasets were used with each representation. For feature selection, the authors applied Information Gain, which employs entropy to measure impurity. Similarly, X. Li et al., 2020 proposed a boosting method that involved Co-Forest, KNN, and Random Tree algorithms as base classifiers. In data pre-processing, they decoded and lowercased the data to obtain its original form. Domain expertise-based

feature extraction method was used to extract 70 features from the traffic flow (HTTP request) and XSS payloads. Feature selection was performed using the CfsSubsetEval feature evaluation method and a greedy search algorithm, resulting in the selection of only 14 features from the original 70 handcrafted features.

The another bagging ensemble learning method was found adopted by (Nagarjun & Ahamad, 2020), where various classifiers have been employed. Data cleaning used the tokenization method. The encoded data was patterned by the neural network and padded to match a fixed-width sequence, accommodating variable lengths in queries or lines of code. This method falls under the category of statistical-based feature extraction. Also in the method suggested by (Y. Zhou & Wang, 2019), a set of Naïve Bayes algorithms were employed as a bagging ensemble learning method. Their learning algorithm incorporated scoring and searching techniques to evaluate the importance of nodes based on their influences. A voting method was also applied to ensemble the individual models to create the ensemble learner. Here, only 30 handcrafted features were directly extracted from XSS payloads using a domain expertise-based method. Tama et al., 2020 implemented a stacking ensemble learning approach, utilizing several classifiers, including RF, Gradient Boost, and XGBoost. Abu Al-Haija, 2023 introduced three distinct models, with two being machine learning-based and the third employing a hybrid ensemble learning technique that combines bagging and boosting. Notably, an encoding method was used to transform the handcrafted features extracted from the payloads, resulting in a total of 7 features. Among the various approaches, the hybrid ensemble learning method emerged as the most effective, delivering the best performance results.

Another trend of ML methods involves the application of RL in the context of cybersecurity, particularly for detecting XSS attacks. RL has been employed in various studies for generating adversarial samples and producing new attack samples. This is achieved by following specific mutation rules to create an adversarial example from existing XSS attack sample. Y. Fang et al., 2019 suggested a DDQN RL-based Adversarial XSS attack generation model. The newly generated samples were used to challenge both Black-box and White-box detection models. The successful bypassed samples were then utilized for retraining the XSS classifier model. Similar approaches were implemented in works by (Q. Wang et al., 2022), who employed the soft Q-learning algorithm, and (L. Chen et al., 2022; Lee et al., 2022), they have utilized the Actor-Critic (SAC) RL algorithm.

Z. Liu et al., 2022 presented a method for generating fuzzing test vectors using genetic algorithms to enhance the accuracy of detecting XSS vulnerabilities in web applications. This method, known as GAXSS, leveraged taint tracking to generate diverse test data and incorporated a modification technique to evade security detection mechanisms in the genetic algorithm's mutation step. Another approach presented by X. Q. Zhang et al., (2020), a GAN-based adversarial training method named MCTS-T was proposed to defend against XSS adversarial examples. The MCTS-T algorithm considered two factors, "modification position" and "modification action," in generating adversarial examples. Bypassing rules, including hexadecimal encoding, decimal encoding, URL encoding, inserting invalid characters, and case mixture, were used to modify existing XSS attacks. The original XSS attack samples and newly generated adversarial data were tested against the suggested CNN-based detection model. While these studies have demonstrated the effectiveness of RL-based

approaches in generating sophisticated adversarial samples and improving the accuracy of XSS detection models, they often neglect the considerations of dynamic detection and feature drift. Finally, in another approach proposed by Tariq et al., (2021), which uses an integrated XSS detection approach based on three models was proposed, namely, A Genetic Algorithm (GA), Statistical Inference, and Reinforcement Learning (RL). First, the best chromosomes for detecting a vulnerability are compared. If differentiation cannot be achieved, fit chromosomes are compared to find the most differentiating chromosomes first. In this phase, a payload will most likely be classified as susceptible or non-vulnerable, but if the payload cannot be distinguished, the next phase of statistical inference will be performed. This RL is used to adapt to new attacks and is accomplished by altering, deleting, and adding chromosomes inside the model. The Distinguisher, Declarator, and Model are the components that the RL module employs to improve performance. However, this work can be criticized for lacking structure for knowledge preservation, such as an incremental learning. In addition, it does not explicitly address the issues of feature drift that are caused by the XSS attack evolution.

Therefore, the method proposed by Tariq et al. (2021) has been selected as the benchmark for evaluating the DQN-MAFS model. This selection is due to Tariq et al.'s adoption of reinforcement learning (RL) for adaptive XSS attack detection by selecting the optimal chromosomes, which consist of the most relevant features, thereby enabling effective XSS attack detection. The comparison will focus on how well each model handles evolving XSS attacks and addresses the challenge of feature drift.

Table 2.3: Literature Survey of the existing ML -based XSS detection approaches

Authors/year	Feature extraction/representation				Feature selection	Feature drift	Classification	Incremental learning
	Domain expertise-based	Statistical-based	NLP-based					
(Alazab et al., 2022)	✓	×	×	✓	✓	×	SVM	×
(Tariq et al., 2021)	✓	✓	×	✓/Embedded		-	Distance based using RL	×
(Mokbal, Dan, Imran, Jiuchuan, et al., 2019)	×	✓	×	×	×	×	MLP	×
(Mokbal et al., 2021)	×	✓	×	✓		-	Multiple classifier	-
(Mokbal et al., 2020)	×	✓	×	×	×	×	XGBoost	×
(Zhou & Wang, 2019)	✓	×	×	×	×	×	Bayesian networks	×
(V. K. Malviya et al., 2021)	×	✓	×	×	×	×	SVM and other classifiers	×
(Lu et al., 2022)	×	✓	×	×	×	×	RF	×
(Panigrahi et al., 2022)	×	×	×	✓		×	Naïve Bays and DT	×
(H.-C. Chen et al., 2021)	×	×	×	×	×	×	RF and other classifiers	×
(Rozi et al., 2022)	×	×	✓	✓		×	XGBoost	×
(Phung & Mimura, 2021)	×	×	✓	×	×	×	SVM	×
(Z. Cheng et al., 2021)	×	×	✓	×	×	×	Set of classifiers	×
(V. Malviya et al., 2018)	×	✓	×	×	×	×	SVM and other classifiers	×
(Ghaleb et al., 2022)	×	✓	✓	✓	×	×	Ensemble learning	×
(X. Li et al., 2020)	✓	×	×	✓	×	×	Ensemble learning	×
(Nagarjun & Ahamad, 2020)	×	✓	×	×	×	×	Ensemble learning	×
(Tama et al., 2020)	×	×	×	×	×	×	Ensemble learning	×
(Y. Fang et al., 2019)	×	×	×	×	×	×	DDQN RL	-
(Q. Wang et al., 2022)	×	×	×	×	×	×	MLP, SVM, and LSTM	×
(L. Chen et al., 2022)	×	×	×	×	×	×	Soft Actor-Critic (SAC) RL	×
(Lee et al., 2022)	×	×	×	×	×	×	Advanced Actor-Critic (SAC) RL	×
(Z. Liu et al., 2022)	×	×	×	×	×	×	GA	×
(Abu Al-Haija, 2023)	✓	✓	×	×	×	×	Ensemble learning	×

Based on reviewed state-of-the-art machine learning (ML) approaches for XSS attack detection in Tabel 2.3, highlights several critical gaps and limitations. While current methods address XSS detection through various aspects such as data cleaning, feature extraction, feature selection, and classifier models, they predominantly rely on domain expertise-based and statistical-based methods for feature extraction. These methods typically involve predefined feature sets derived from domain knowledge or statistical analysis, represented numerically to aid in detecting XSS attacks. While effective in certain scenarios, these approaches are inherently static, limiting their ability to adapt to the evolving nature of XSS attacks. Additionally, feature selection in these methods often employs techniques like selection-based knowledge, filtering, wrapper, embedded methods, and attention mechanisms. Although these techniques improve model performance, they largely ignore the critical issue of feature drift—where the relevance of features changes over time due to new evasion strategies by attackers—posing a significant challenge that current ML-based methods are ill-equipped to handle.

Another significant limitation is the lack of incremental learning in these approaches. Incremental learning, essential for continuously updating models as new data arrives, is crucial for detecting new XSS attack variants. The absence of incremental learning capabilities in existing methods is likely due to the recent emergence of concepts like feature drift and incremental learning in cybersecurity. Consequently, most models are designed for static learning environments, rendering them less effective in real-world scenarios where data and attack vectors are constantly evolving. To address these gaps, we propose the DQN-MAFS model, an adaptive and dynamic approach to feature selection that explicitly manages feature drift. By integrating multi-agent

reinforcement learning (RL) and supporting incremental learning, our model dynamically selects relevant features and adapts to changing data characteristics over time. This approach offers a more resilient and adaptable solution for XSS attack detection, marking a significant improvement over traditional methods. Our proposed method will be rigorously compared with existing work, such as that of Tariq et al. (2021), to demonstrate its effectiveness in overcoming the challenges of feature drift and incremental learning in XSS detection.

2.7.1.2 Deep Learning-based XSS Attack Detection Methods

Within the realm of XSS attack detection, deep learning techniques have gained popularity, with researchers have adopted various DL algorithms such as CNN, LSTM, RNN, Bi-LSTM, Bi-RNN, and autoencoder. As we delve into the literature, it becomes evident that two principal categories of XSS detection have highlighted, each based on specific XSS data types: Payload-based and JS-file-based (as elaborated in Section 2.3.2). In this sub-section, we thoroughly examine how deep learning methodologies are applied to address XSS attack detection. Table 2.4 presents the analysis of DL methods adopted in the literature.

For Payload-based XSS attack detection methods, some studies were based on CNN as a means of learning and feature extraction. For instance, Maurel et al., (2022) conducted an experimental comparison between static approaches and deep neural networks. However, despite experimenting with various hyper-parameters, their method suffered from significant drawbacks, particularly in terms of high false positive and false negative rates. Similarly, Kumar et al., (2022) leveraged CNN for

both feature extraction via feature map layers and feature reduction through pooling layers, emphasizing the efficiency of CNN in processing complex data representations. In another approach, Chaudhary et al., 2021 applied CNN following data preparation processes involving decoding and contextual tagging in Fog nodes connected to an IoT network, showcasing the adaptability of CNN in different computational environments. Additionally, Abaimov & Bianchi, (2019) introduced the CODDLE model for identifying injection attacks, including XSS. Their method combined statistical-based feature extraction with encoding techniques to transform original data inputs to numeric tokens, which were subsequently fed into CNN for effective learning and detection.

Melicher et al., (2021) adopted a continuous Bag of Words (CBoW) model with domain expertise effectively represent the functions keywords of JS. They collected data through browser-based taint tracking. Utilizing this data, they trained a DNN to predict vulnerabilities in JS function. This approach yielded a classification accuracy of 94.5%. In a similar vein, Kuppa et al., (2022) introduced a novel approach for detecting malicious code in sustainable smart cities called ConvXSS. The pre-processing phase involves decoding and generalizing code lines, semantically labelling words, and converting labels to binary numbers in which only 50 features were extracted. Code snippets are represented using ASCII values and transformed into an image-like format suitable for CNNs model that optimized through hyper-parameter tuning. However, despite the effectiveness of these methods in detection, they were based on batch learning, which assumes a static nature of XSS attacks, and they also suffered from the challenges of high dimensionality and complexity.

Niu & Li, (2020) proposed a hybrid approach that leverages a CNN-GRU model for web attack detection, where the GRU layers capture the serialization relationship of context within web event sentences. This method combined Word2Vec with statistical-based feature extraction to convert the most frequent sensitive keywords into vector representations, ultimately yielding eight features. Similarly, Akaishi et al., (2019) combined Word2Vec with CNN for feature learning and using Naïve Bayes for classification. Another work conducted by (H. Yan et al., 2022) adopted a data cleaning process comprising normalization, decoding, and tokenization. Then, they employed pre-trained CNN-based Residual Network (ResNet) models, where the convolutional layers of ResNet were responsible for extracting features from the Word2Vec embeddings of input text data. J. Yang et al., (2020) introduced a method by utilizing tokenization method combined with a single layer of Bi-LSTM networks to encode features and capture semantics at both word and segment levels. An attention mechanism was incorporated to enhance this process, with the extracted features subsequently fed into a CNN-based text classification model, TextCNN, for further learning and detection. Additionally, G. Zhang et al., (2019) adopted a hybrid data cleaning approach that merged normalization and tokenization techniques. The feature extraction process employs Word2Vec in conjunction with CNNs to effectively capture local patterns and extract meaningful semantic features from HTML tags and DOM methods.

Hu et al., (2023) introduced a detection model incorporating a two-channel multi-scale feature fusion mechanism within a self-attention method. This model begins by mapping XSS payloads into spatial vectors through data preprocessing utilizing Word2Vec. Subsequently, it conducts feature extraction through two-channels. First

channel focuses on extracting local features from XSS payloads at different scales by employing three parallel one-dimensional convolutional layers with distinct kernel sizes (2, 4, 6). The second channel focusses on extracting semantic information of payloads from both positive and negative directions using a single layer of bi-LSTM network. A self-attention mechanism is then embedded to enhance these semantic features. Finally, the extracted local features with semantic features by both channels are fused and used for detection. Similarly, W. Yang et al., (2019) proposed a Convolutional Gated Recurrent Unit (CGRU) neural network for detecting XSS and SQL detection. This approach utilizes characters as text classification features and introduces a feature representation method based on malicious keywords specific to URLs. The model is divided into three parts: (1) Keyword-Based URL Character Embedding, where the original URL characters are mapped into a low-dimensional vector, (2) Feature Extraction Module, where the convolutional neural network extracts features, with the GRU serving as a pooling layer, (3) Classification Module, where a fully connected neural network for classifying the detected features.

Tekerek, (2021) suggested the data cleaning process that involves solely the utilization of the normalization method. The approach includes the adoption of the BoW technique in conjunction with a statistical feature extraction method. This method involves counting the frequency of BoW vectors, where each word count is treated as an individual feature. To facilitate learning and classification, a CNN is employed. Highest classification accuracy and F1-score obtained by this method are 97.07% and 97.51%, respectively. Likewise, Shahid et al., (2022), no specific data cleaning methods were employed. The authors utilized a statistical-based feature extraction technique that relied on predefined features combined with the CNN algorithm for the learning process.

On the other side, for JS file based methods, similar to the first categories some methods have used CNN, as those demonstrated by (R. Yan et al., 2018), (Zhang et al., 2022). R. Yan et al., (2018), suggested a hybrid deep learning network (HDLN) that combines convolutional layers, recurrent layers (Long Short-Term Memory layers), and fully connected layers. At first, it extracts more features from Abstract Syntax Tree (AST) of JavaScript by using tri-gram model with sliding window length. Then three different feature selection methods namely, information gain, Chi-square test and document frequency, were employed to select key features. After that, these feature vectors were used to train HDLN to distinguish vulnerable applications from normal ones. Additionally, Z. Zhang et al., (2022) introduced an adaptive code length deep learning network called JACNet was proposed to detect JavaScript malicious code. The JACNet network is composed of a CNN-based model called a deep residual convolution network RDCNet, single layer of BiLSTM, and Transformer to capture the association features of variable distance between codes. The authors firstly design an abstract syntax tree recombination algorithm to provide rich syntax information for feature extraction. Secondly, they use a RDCNet to capture short-distance association features between codes. The study also uses Doc2vec to generate the word vector and serves as the embedding layer of the model word vector. However, these methods still suffer from significant limitations, particularly in their failure to incorporate feature reduction techniques and their infrequent updates, which hinders their ability to detect emerging variants of attacks. Without effective feature reduction, the methods may be inefficient, processing excessive amounts of irrelevant data, which can reduce accuracy and increase computational costs. Additionally, the lack of regular updates means these methods are unable to adapt to new or evolving threats, making them less effective in dynamic environments where attack patterns continuously change.

Other methods used GCN with XSS payload, such as (Z. Liu et al., 2021; H. Pan et al., 2022). Z. Liu et al., (2021) proposed a detection model based on graph convolutional networks. After the samples are pre-processed and constructed, the processed data are converted to a graph structure. This is done by treating each piece of payload data and a word content as a point, then connecting these points according to the rules to construct a graph. First of all, a connection is established between each piece of data and its corresponding word and consideration an adjacency relation. The resulted graph is fed into the graph convolutional network and the residual network to train the cross-site scripting detection model. Additionally, H. Pan et al., (2022), adopted a comprehensive approach involving three distinct data cleaning methods. Their proposed methodology encompasses the fusion of Word2Vec (CBOW) and Convolutional-based feature extraction (GCN) techniques for the purpose of feature extraction.

On the other side, some methods have used GCN with JS data (Y. Fang et al., 2022; Z. Liu et al., 2023b). Z. Liu et al., (2023b) proposed a multi-feature fusion-based neural network vulnerability detection model namely as a MFXSS. It combines two features extraction methods are Graph feature extraction and code string features extraction. The first one represents to source code as a graph structure by AST and CFG, and each line of code is used as a point. The code is vectorized and embedded by the word vector model, and the graph features of the source code are extracted by the graph convolutional neural network. In other hand, the second method tracks the sensitive methods or attributes, the code execution sequence for extraction. This achieved by using Word2Vec and single layer of Bi-LSTM network with attention mechanism. Finally, the fused feature vectors are used to detect and predict XSS vulnerabilities in

JavaScript. The method introduced by (Y. Fang et al., 2022), proposes a static detection model called JStrong for detecting malicious JavaScript based on graph neural network (GCN). The model generates an AST from the JavaScript source code and adds data flow and control flow information into the program dependency graph. The nodes and edges of the graph are then embedded into a feature vector and the features of the whole graph are fully learned through the graph neural network. The study also includes a new graph pruning algorithm to extract and retain complicated high-level semantic features and compares the performance of various code graph representation methods in JavaScript and their influence on the graph neural network classifier. However, only focuses on detecting XSS attacks in JavaScript code, it requires a significant amount of computational resources to generate the program dependency graph and perform the graph neural network classification, and it may not be effective in detecting new or previously unseen types of XSS attacks. Additionally, the graph pruning algorithm used to extract high-level semantic features may not be able to fully capture all the important information in the code, leading to potential false negatives in the detection process.

The usage of standalone standard LSTM with payload is found adopted by (Y. Fang, Li, et al., 2018; Stiawan et al., 2023). Y. Fang, Li, et al., (2018) presented a deep learning approach for detecting XSS attacks called DeepXSS. The approach consists of pre-processing the input data, including decoding, generalization, and tokenization, and extracting features using word2vec. The features are then fed into a single layer of traditional (LSTM) recurrent neural network to train a classifier that can distinguish XSS samples from benign samples. In the same context, Y. Fang, Xu, et al., (2020) proposed a model for email XSS attack detection using Bidirectional-RNN algorithm

and the Attention mechanism. Word2Vec model was used for pre-processing the email information and fed them for the classification model. In the work conducted by (Stiawan et al., 2023), the data cleaning process exclusively involved normalization. Contextual analysis for feature learning the Long-short Term Memory (LSTM) algorithm was utilized. Additionally, Principal Component Analysis (PCA) was employed as a means of feature reduction.

The usage of RNN with JS files is found employed by (S. Mohammadi & Namadchian, 2020; Phung & Mimura, 2021), while standard LSTM is found adopted by (Fang et al., 2018). In the work of (Phung & Mimura, 2021), an oversampling method are used to balance the dataset. In the data pre-processing step, pattern replacement was used. The dataset was then passed to the Doc2Vec or the Attention model for feature extraction. The extracted feature was used to train the SVM classifier for prediction. The attention model used in this study has 2 Bi-directional RNN (2 LSTM) layers for prediction tasks, The hidden states from each RNN layer are concatenated and then a weighted sum of the encoded states is used to make predictions. The last layer shows the confidence level and is connected to an output node. Similarly, S. Mohammadi & Namadchian, (2020) also proposed RNN-based method, in which attention-based RNN network is used to encode the request-response data after transforming them into a numerical representation by using Word2Vec model. Furthermore, based on the requests, the value of attention is produced. To reflect the response, these values are sent through a decoder RNN network, and the output is fed into a fully-connected layer for classification. Y. Fang, Huang, et al., (2018) suggested a malicious JavaScript detection model based on standard LSTM, which extracts features from the semantic level of bytecode and optimizes the method of word vector. The study uses the V8

engine to extract bytecode sequences, but because the DOM and BOM objects in the browser environment cannot be identified, it uses the jsdom package library of Nodejs for simulation. The study presents word-vector extraction of bytecode based on the word2vec model, and uses the TF-IDF algorithm with weighting mechanism to feature selection and the rejection of invalid high-frequency words and keep the effective high-frequency words. RNNs and LSTMs, while effective for sequence modeling, are limited by their inability to handle long-term dependencies efficiently and suffer from high computational costs due to their sequential nature, which hampers parallel processing and scalability.

To extract deep semantic information from the JS context, (X. Y. Song et al., 2020) employed single BI-LSTM layer, whereas (Y. Huang et al., 2021) used more advanced method with two layers BI-LSTM. X. Y. Song et al., (2020) proposed a deep learning-based method for detecting malicious JavaScript including XSS and other attacks. The approach starts with de-obfuscating the raw JavaScript code to uncover its semantic information. This information is then used to generate the Program Dependency Graph (PDG), which is constructed based on the Abstract Syntax Tree (AST) and the Control Flow Graph (CFG). The PDG is used to identify dependencies between statements and facilitate program slicing to transform into vectors. The vectors are then fed into a single layer of Bidirectional Long Short-Term Memory (BLSTM) neural network for malicious JavaScript detection. The authors also map user-defined variables and functions to symbolic names, divide the code into a token sequence using lexical analysis, and use the word2vec algorithm as an embedding tool to encode each token into a vector. On the other hand, Y. Huang et al., (2021) proposed JSContana,, which aims to enhance detection through adaptable context analysis and efficient key feature

extraction to extract the real contextual information of the code. The method consists of three parts: data pre-processing, feature learning, and classification. In data pre-processing, JS files are parsed into an AST and transformed into syntax unit sequences with detailed information. In feature learning, dynamic word embeddings are introduced using the Word2Vec and two Bi-LSTM layers network, which obtain dynamic word vectors through context analysis of syntax unit sequences. In the classification step, dynamic word vectors are fed into the TextCNN model classification based on key feature extraction.

The sole study we came across that employs an autoencoder technique in conjunction with XSS payloads is demonstrated by (Y. Pan et al., 2019). In this study, no specific data cleaning method was applied. Feature reduction was conducted using Principal Component Analysis (PCA). The utilization of an autoencoder for XSS attack detection revolved around encoding and reconstructing the call graph for comprehensive deep learning. To achieve this, a stacked denoising autoencoder was trained with the aid of the Robust Software Modelling Tool (RMST), a dynamic analysis tool that observes code behaviour during runtime. By leveraging a condensed representation of the original features with unlabeled request data, the autoencoder identifies anomalies through the computation of the reconstruction error associated with the request data.

Table 2.4: Literature Survey of the existing works in XSS detection using deep learning models

Authors/year	DL method	Feature extraction/representation						Feature selection	Feature drift	Classification	Incremental learning
		Domain expertise-based	Statistical-based	NLP-based	Contextual analysis	Convolutional-based					
(Maurel et al., 2022)	DNN	×	×	✓	×	×	×	Attention	×	Sigmoid	×
(Kumar et al., 2022)	CNN	×	×	×	×	✓		✓	×	Softmax	×
(Y. Huang et al., 2021)	CNN-based model (TextCNN)	×	×	✓	✓	×		×	×	Softmax	×
(Z. Liu et al., 2021)	GCN and Residual network block)	×	×	✓	×	✓		×	×	Softmax	×
(Y. Fang et al., 2018)	LSTM	×	×	✓	✓	×		×	×	Softmax	×
(Abaimov & Bianchi, 2019)	CNN	×	✓	×	×	✓		×	×	Softmax	×
(Chaudhary et al., 2021)	CNN	×	✓	×	×	✓		PCA	×	Sigmoid	×
(Y. Fang, Xu, et al., 2020)	Bi-RNN	×	×	✓	×	✓		Attention	×	Softmax	×
(J. Yang et al., 2020)	TextCNN	×	×	✓	✓	×		Attention	×	Softmax	×
(X. Q. Zhang et al., 2020)	DL	×	×	×	×	×		×	×	softmax	×
(Hu et al., 2023)	CNN and Bi-LSTM	×	×	✓	✓	✓		Self-attention	×	softmax	×
(Lei et al., 2020)	LSTM	×	×	✓	✓	×		×	×	softmax	×
(W. Yang et al., 2019)	CNN and GRU	×	×	×	✓	✓		×	×	softmax	×
(R. Yan et al., 2018)	CNN and LSTM	×	×	✓	✓	✓		✓	×	softmax	×
(Stiawan et al., 2023)	LSTM	×	×	×	✓	×		PCA	×	Softmax	×
(G. Zhang et al., 2019)	CNN	×	×	✓	×	✓		×	×	softmax	×
(Shahid et al., 2022)	CNN	×	✓	×	×	✓		×	×	softmax	×

Table 2.4: Continued

(Z. Zhang et al., 2022)	Residual convolution network	×	×	✓	✓	×	×	×
(Z. Liu et al., 2023b)	GCN, MLP, and Bi-RNN	×	×	✓	✓	×	Self-attention	×
(Y. Fang et al., 2022).	GCN	×	×	✓	×	×	×	×
(X. Y. Song et al., 2020)	Bi-LSTM	×	×	×	✓	×	×	×
(S. Mohammadi & Namadchian, 2020)	RNN	×	×	✓	×	×	Attention	×
(Melicher et al., 2021)	DNN	✓	×	✓	×	×	×	×
(Kuppa et al., 2022)	DL	×	×	✓	×	×	×	×
(Niu & Li, 2020)	CNN and GRU	×	✓	✓	×	×	×	×
(Akaishi et al., 2019)	CNN	×	✓	✓	×	×	×	×
(H. Yan et al., 2022)	CNN-based model (ResNet)	×	×	✓	×	×	×	×
(H. Pan et al., 2022)	GCN	×	×	✓	×	×	PCA	×
(Y. Pan et al., 2019)	Autoencoder	×	×	✓	×	×	-	×
(Tekerek, 2021)	CNN	×	✓	✓	×	×	Sigmoid	×

In summary, current state-of-the-art XSS attack detection methods using deep learning (DL) approaches have predominantly focused on various aspects, including data cleaning, feature extraction, feature selection, and classifier models, as shown in Table 2.4. The feature extraction techniques employed in these methods primarily leverage natural language processing (NLP) models, contextual analysis, and convolutional approaches. For instance, methods such as (Kuppa et al., 2022; S. Mohammadi & Namadchian, 2020), Doc2Vec (Ngoc & Mimura, 2021; Phung & Mimura, 2021), Code2Vec (Alon et al., 2019), (Akaishi et al., 2019; Niu & Li, 2020) have been extensively used for NLP-based feature extraction. Contextual analysis techniques, including Recurrent Neural Networks (RNN) (Qin et al., 2018; Stiawan et al., 2023), Bi-directional RNN (Bi-RNN) (Y. Fang, Xu, et al., 2020), Long Short-Term Memory (LSTM) (Stiawan et al., 2023), and Bi-LSTM, have been adopted to explore deep semantic features in XSS attack detection, as demonstrated in works by (Y. Fang, Li, et al., 2018; Lei et al., 2020; Yu et al., 2018). Convolutional-based methods, such as CNN and Graph Convolutional Networks (GCN), have also been employed to capture hierarchical patterns in textual data, with notable examples including studies by (Hu et al., 2023; Z. Liu et al., 2021, 2023b; H. Pan et al., 2022; H. Yan et al., 2022; R. Yan et al., 2018; G. Zhang et al., 2019).

Despite these advancements, several critical issues persist within the current DL-based approaches. Notably, feature selection has often been overlooked or handled sub-optimally. While attention mechanisms have been employed in some studies for feature selection with NLP approaches, such as those by (Y. Fang, Xu, et al., 2020; Hu et al., 2023; Z. Liu et al., 2023b; Maurel et al., 2022; S. Mohammadi & Namadchian, 2020; J. Yang et al., 2020), the other works have either relied on basic dimensionality

reduction techniques like PCA (Chaudhary et al., 2021; Y. Pan et al., 2019; Stiawan et al., 2023) or neglected the dynamic nature of feature relevance altogether. This oversight is particularly problematic given the evolving nature of XSS attacks, where feature drift over time is a significant challenge. Moreover, existing methods are primarily based on batch learning, which assumes that the entire dataset is available upfront, and thus, fail to address the need for incremental learning and dynamic feature adaptation. Additionally, the high complexity of textual data, coupled with the costly training requirements of DL models with large dimensions, further exacerbates the limitations of these methods in effectively handling real-world XSS attack scenarios. To address these gaps, this research proposes the 2TS-DFS model, which introduces an adaptive and dynamic approach to feature selection that explicitly handles feature drift both spatially and temporally. Unlike previous methods, such as the dynamic word embedding approach by Huang et al. (2021), which aimed to capture dynamic feature representation but struggled with high computational complexity, the 2TS-DFS model is designed to efficiently manage feature drift in a more granular and computationally feasible manner. By dynamically selecting features in real-time as new data arrives, this model aims to significantly improve the robustness and accuracy of XSS attack detection in rapidly evolving environments.

2.7.2 RL For Dynamic Feature Selection (DFS)

In the realm of handling feature drift, Only a few methods are found in the literature that specifically were proposed to such problem (Barddal et al., 2019; X. Wang et al., 2022). A noteworthy approach for handling feature drift is the Dynamic Feature Selection (DFS) that selectively focuses on the most relevant features while

disregarding irrelevant ones (Barddal et al., 2016; Ferone & Maratea, 2021; Villa-Blanco et al., 2023). Although, previous methods (traditional feature selection methods as explained in Section 2.6) often assumed a constant underlying distribution over time, recent trends in the literature have highlighted the effectiveness of using Reinforcement Learning (RL) for explicit DFS, as RL excels in handling dynamic changes and updating the knowledge embedded within RL agents. Various types of agents have been employed in this context as shown in Table 2.5.

Traditional Q-learning agent has a prominent choice in many studies. For instance, the work proposed by (Paniri et al., 2021), introduces a feature selection method based on Ant Colony Optimization (ACO) that utilizes the temporal difference (TD) reinforcement learning algorithm for heuristic learning. Features correspond to states (S), and the ants' selection of unvisited features represents their actions, with reward value based on their taken actions. In another study, R. Xu et al., (2021) proposed a method that fits a discriminant function using Q-learning. The agent selects classification actions while approximating the function based on agent decisions' feedback. In alternative approach was suggest by Covert et al., (2023), utilizes a single-agent RL technique based on Q-learning, where the state represents the feature set, and the policy employs a greedy approach to select or deselect features based on their conditional-mutual-information (CMI). This approach directly predicts the best feature selection and differs from previous work by estimating CMI. However, these algorithms were built based on a linear discriminant function which limiting it to simple data. M. Kim et al., (2022) proposed DFS method by utilizing MARL. This approach involves main agents, guide agents, state, environment, actions, and different kinds of rewards values to facilitate DFS. Main agent acts as a feature selector that

learns through rewards, while guide agent evaluates main agents' behavior. Rewards values are measured based on classification accuracy.

Another agent type is the deep deterministic policy gradient (DDPG), that has been employed by Y. Cheng et al., (2021). This method modeled the feature selection process as a Markov Decision Process (MDP) and formalized it as an RL problem. They proposed a DRL-based Feature-Selector (DRLFS), incorporating dynamic randomness policies for exploration and exploitation control, along with two search protocols: fixed depth search and adaptive depth search.

Other researchers have adopted deep Q-learning network agents (DQN). For instance, K. Liu et al., (2021) introduced a multi-agent framework for DFS where each single agent corresponds to one feature. The final selected subset features are determined based on the collective decisions of all agents. The agents rely on state representation that is built using descriptive statistics; and the reward schema encompasses relevancy, redundancy, and accuracy factors, weighted appropriately. DQN network is employed and trained using experience replay buffer sampling via Gaussian Mixture Models (GMM). However, this approach lacks a distinct mechanism for reward distribution among features based on their actual contribution, and not apply a specific mechanism for balancing exploration and exploitation processes. To further highlight the effectiveness of the proposed approach, this study is selected as a benchmark to evaluate the performance of the proposed DQN-MAFS model. The DQN-MAFS model introduces a novel fair reward distribution method that calculates reward values for each agent based on their genuine contribution to selecting the most optimal features, thereby enhancing the detection performance. Unlike the original study,

which did not support incremental learning, our proposed model has been designed to handle the dynamic nature of feature selection in an evolving environment by supporting incremental learning. To ensure a fair comparison, the original method has been re-implemented with the same configuration design as our work, but now incorporates incremental learning techniques. The models were evaluated using datasets with varying feature sizes, further demonstrating the superior performance of our proposed reward distribution method and its ability to adapt to the evolving landscape of XSS attack detection.

In another study of (Fan et al., 2020), proposed a navigation mechanism using interactive-reinforcement learning (IRL) with decision-tree feedback (DTF) as an external trainer-agent for learning feedback. A GCN is used to learn state representation from both the tree and the graph simultaneously, with customized reward assignment for agents based on the features relevancy of decision tree. Similarly, the work of (Z. Fang et al., 2019), introduced a framework of a Deep Q-learning based Feature-Selection (DQFSA) for malware detection. The Q-learning agent operates in a sequential learning manner. The reward value is determined based on the accuracy of the adopted machine learning classifier. X. Li et al., (2021) proposed a DQN approach where a single agent is represented by DQN for taking one action regarding one feature at a time. The reward is structured using a piecewise equation combined with four branches. Additionally, Wu et al., (2022) suggested a framework using Double-DQN (DDQN) algorithm for malware classification. The authors adopted RNN as the decision network of DDQN to accommodate input variable length utilizing word embedding for feature representation to uncover semantic relevance. Finally, Fan et al., (2021) presented a framework for Group-based

Interactive-Reinforced Feature-Selection (GIRFS) that balances single-agent and multi-agent RFS. They begin with grouping features based on feature similarity and create agents for each group. Additionally, they proposed a hierarchical teacher-like trainer to offer external action advice to agents.

The reviewed methods are analyzed in Table 2.5. Overall, it is evident that each employs a variety of RL elements, including state, action, reward, agent type, and exploration vs. exploitation balance. However, none of these existing methods has provided a comprehensive algorithm for reward distribution in the context of multi-agent rewarding. The challenge lies in determining the unequal contributions of agents to overall performance (Gronauer & Diepold, 2022; T. T. Nguyen et al., 2020). Hence, it is necessary to come up with an approach that accomplishes fair reward distribution based on the real contribution of each agent.

Moreover, spatial and temporal feature relevance is another critical dimension that has been largely overlooked by traditional methods. In real-world applications, the relevance of features can not only change over time (temporal drift) but can also vary within the same time interval (spatial drift), depending on the structural similarities among data samples. For example, two XSS payloads received within the same time frame might have different underlying structures, necessitating different features for effective detection. Traditional methods (traditional feature selection methods as explained in Section 2.6) that operate on the assumption of uniform feature relevance across all samples in a given batch or time chunk fail to capture this complexity, leading to suboptimal detection performance.

Table 2.5: Literature Survey of the existing algorithms in RL based dynamic feature selection

Article	MARL	State	Action	Reward	Reward distribution	Agent	Exploration Exploitation Balance	Classifier
(Y. Cheng et al., 2021)	×	Encoding of the subset of selected features	Binary decision	Decline of Error	×	DDPG	Randomness policy	SVM
(K. Liu et al., 2021)	✓	Descriptive statistics	Binary decision	Accuracy, redundancy, relevance	×	DQN	ϵ -greedy	RF, LASSO, DT, SVM, XGBoost
(Fan et al., 2020)	✓	GCN	Binary decision	DT based personal rewarding and Accuracy from hybrid teaching	×	DQN	ϵ -greedy	Decision tree
(M. Kim et al., 2022)	✓		Binary decision	Classification accuracy	×	Q-learning	ϵ -greedy	ANN
(Z. Fang et al., 2019)	✓	Selected feature	Select/deselect	Classification accuracy	×	DQN	ϵ -greedy	Set of classifiers
(Paniri et al., 2021)	✓	Feature	Select/deselect	Cos similarity and correlation	×	Q-learning	Polynomial decay rate	KNN
(R. Xu et al., 2021)	×	Feature	Class	Fixed values for rewarding and punishment	×	Q-learning	ϵ -greedy	Linear discriminant function
(Fan et al., 2021)	✓	Graph Convolutional Network (GCN)	Selection /de-selection	Classification accuracy	×	N/A	ϵ -greedy	Decision tree
(X. Li et al., 2021)	×	Binary	Selection /de-selection	Piecewise based on the accuracy	×	DQN	ϵ -greedy	KNN
(Wu et al., 2022)	×	Binary	sequential selection by RNN	Classification accuracy	×	DDQN	ϵ -greedy	Set of classifiers
(Covert et al., 2023)	×	Binary mask	Select/deselect	Prediction loss value	×	Q-learning	greedy	Bayes classifier
Proposed approach	✓	Binary	Binary decision	Developed (FARD-DFS)	✓	DQN	Decay ϵ greedy	Set of classifiers

In the subsequent section, we delve into the existing challenges associated with the current employment of ML and DL-based techniques for the detection of XSS attacks and RL-based dynamic feature selection.

2.8 Open Challenges and Gap Discussion

The presented literature delves into the existing methods have been employed for XSS attack detection within the context of web applications. Such methods that based traditional ML and DL techniques, has proven inadequate in effectively tackling the constantly evolved XSS attacks. This inadequacy is particularly evident when it comes to detecting minor variants like unknown attacks, which tend to evade conventional detection methods. Moreover, the absence of a comprehensive approach for addressing attack drifts and the evolving behaviour inherent to XSS attacks has surfaced as a major gap in the field.

Throughout the literature survey, the focus of existing ML/DL-based XSS attack detection methods have predominantly been on feature extraction, reduction, classification, and data generation. However, the dynamics of data, evolving attacks, and the challenge of continuous knowledge updates in the face of novel attacks have not been systematically addressed. Because the existing ML/DL-based XSS attack detection method primarily employ offline learning and batch learning methods that consider the static distribution of data. This ignores the consideration of data dynamic aspects can led to a disconnect between the current methodologies and the evolving of XSS attacks. Notably, the category of evolved XSS attacks, characterized by new malicious payloads with distinct components. Surprisingly, the incremental learning

approach, essential for addressing such evolving challenges, has not been extensively explored in the existing literature.

Among the critical facets that have been largely unexplored are feature drift and the influence of temporal changes on feature relevance. Traditional feature selection methods, while effective for static scenarios, fall short when it comes to handling dynamic relevancy changes. The emergence of feature drift, particularly in the context of XSS attacks, necessitates an approach capable of dynamically adapting to these changes. Reinforcement learning (RL) emerges as a powerful candidate to address dynamic feature selection, enabling models to adapt to evolving feature relevancy. However, existing methods using RL lack a novel approach to fairly distribute rewards among agents, which directly impacts their contribution to overall performance.

Another gap lies in the realm of text representation, where current word embedding methods have been predominantly employed. However, these methods struggle to cope with the rapidly evolving nature of XSS attacks. The limited attention given to the high dimensionality of extracted features and the oversight of stream-based data processing further accentuate the need for a more holistic approach.

Overall, the current state of research fails to provide comprehensive solutions for dynamic XSS attack detection. While some efforts have been made, the evolving nature of attacks requires a dynamic approach, including incremental learning, dynamic feature selection, and context-aware feature extraction. Bridging these gaps demands the creation of models capable of accommodating different data types, adapting to evolving scenarios, and employing suitable methods for handling temporal

and spatial feature changes, thereby ensuring the efficacy of XSS attack detection systems in the ever-changing cyber threat landscape.

2.9 Preliminary Background of the Proposed Models' implementation

As the practicality of machine learning algorithms and models has increased, there is a surge of interest in their application for cyber security purposes (Geetha & Thilagam, 2021). Thus, in this section, we delve into the rich landscape of machine learning (ML) and deep learning (DL) methods that underline our proposed methodology. Within the realm of ML, we harness the power of three distinct algorithms: Decision Trees (DT), Support Vector Machines (SVM), and Reinforcement Learning (RL). Our proposed approach, the DQN-MAFS model, leverages RL for temporal dynamic feature selection. This temporal dynamic selection method equips us to tackle the ever-evolving nature of XSS attacks by adapting to changes in feature relevance over time. Additionally, SVM plays an implicit yet crucial role within our reward schema, complementing RL's capabilities. In addition, DT serves as a detector classifier, effectively identifying XSS attacks based on features selected by RL. In the context of our 2TS-DFS model, we incorporate hierarchical clustering, specifically the Agglomerative algorithm, as an unsupervised ML technique. This algorithm skillfully groups samples based on their similarities, enabling a comprehensive understanding of data spatial patterns. Furthermore, deep learning algorithms were used including Bi-directional Long Short-Term Memory networks (Bi-LSTM) and Convolutional Neural Networks (CNN) that take main role for facilitating feature extraction, representation, and learning. The subsequent sections provide a thorough background on these algorithms and their peers, establishing the foundation upon which our methodology is built.

2.9.1 Decision Trees (DTs)

DTs is a popular classification method used in supervised learning and widely used in various domains. The characteristic tree-like structure of DT models, displaying their choices and corresponding outputs, facilitates their interpretation and implementation as a classification method. DTs utilize packet features during training to determine the packet's class. The objective is to construct the best DT model that captures as much information as possible while minimizing the number of levels in the tree. However, a drawback of DTs is their lack of robustness; minor alterations to the training data can result in significantly different DTs (Cieslak & Chawla, 2008). Furthermore, the information gain metric in DTs tends to favor multi-level features and may be affected by sampling errors, yielding locally optimal solutions rather than globally ideal outcomes across all contexts (Patel & Prajapati, 2018). Various algorithms, such as ID3, C4.5, and CART, have been proposed to generate optimal trees, employing different metrics like information gain (entropy), information gain ratio, and Gini impurity to measure DTs' performance. Mathematically, the Entropy is represented in Equation (2.3) (Frank et al., 2017):

$$Entropy(s) = -\sum_{c \in C} p(c) \log_2 P(c) \quad (2.3)$$

Where s denotes the entropy of the data, c represents the class label of the data, and $P(c)$ denotes the proportion of data points belonging to class label c relative to the total number of the total data points in set s . The entropy values lie between 0 and 1, reflecting the level of uncertainty or disorder in the dataset.

Information gain, on the other hand, measures the discrepancy in entropy before and after a split based on a specific attribute. The attribute that yields the highest information gain will lead to the most effective split, as it demonstrates superior performance in classifying the training data based on its target classification. The standard formula for information gain is often expressed in Equation (2.4) (Frank et al., 2017):

$$\text{Information Gain}(s, a) = \text{Entropy}(s) - \sum_{v \in \text{values}(a)} \frac{|s_v|}{|s|} \text{Entropy}(s_v) \quad (2.4)$$

Where a denotes a particular attribute or class label, $\text{Entropy}(s)$ represents the entropy of dataset s . $\frac{|s_v|}{|s|}$ signifies the proportion of the values in s_v , to the total number of values in dataset s .

Gini impurity represents the likelihood of misclassifying a randomly selected data point in the dataset if it were labelled based on the class distribution of the dataset. Like entropy, if a set s is pure, meaning it belongs to only one class, its impurity is zero. The Gini impurity is calculated using Equation (2.5) (Frank et al., 2017):

$$\text{Gini Impurity} = 1 - \sum_i (p_i)^2 \quad (2.5)$$

Where p_i denotes the probability of a sample being classified under class i .

While Decision Trees (DT) offer an intuitive and powerful approach to classification, their application in detecting XSS attacks presents several challenges that merit closer examination. Despite their advantages—such as interpretability, ease of implementation, and effectiveness in handling both categorical and continuous data—

DTs can suffer from issues like overfitting (Nanfack et al., 2022), particularly when dealing with high-dimensional data or complex attack patterns characteristic of evolving XSS threats. This is a significant limitation, as XSS attacks are increasingly sophisticated and can exploit nuanced vulnerabilities that may not be fully captured by a static DT model.

In comparison to other machine learning methods, such as ensemble techniques (e.g., Random Forests or Gradient Boosted Trees) or deep learning models, DTs may struggle with generalization and robustness, particularly in dynamic environments where the nature of XSS attacks can change over time (Alhamyani & Alshammari, 2024). This raises concerns about the reliability of DT-based models in real-world scenarios where adaptability and precision are critical. Moreover, existing works that utilize DTs for XSS detection often do so in a relatively straightforward manner as implemented by (Panigrahi et al., 2022), without leveraging the full potential of more advanced techniques that could mitigate these limitations, such as integrating DTs with reinforcement learning for dynamic feature selection or employing hybrid models that combine DTs with neural networks.

We select DT as the main classifier and consider it as the detector in the proposed DQN-MAFS model, as explained in Chapter 4 (Section 4.2.4). This model will be updated over time incrementally with new optimal relevant features whenever new labeled samples become available. In other words, the DQN-MAFS supports incremental learning, where all data is not available at once; instead, data arrives sequentially, and accordingly, the detector is updated. Thus, the knowledge of the DT is continually enhanced, enabling it to detect new variants of attacks that may be

generated using different evasion strategies. This approach contrasts with the reviewed methods in Section 2.7 (Table 2.3 and 2.4), which failed to detect such attacks due to their inability to adapt to evolving attack vectors.

Therefore, while DTs provide a foundational method for initial analysis, they may need to be enhanced or combined with other approaches to effectively address the complexities of XSS detection. The proposed model in the subsequent chapters builds on this understanding by integrating DTs within a broader framework that incorporates dynamic feature selection and temporal-spatial analysis, offering a more robust and adaptable solution to the evolving landscape of XSS attacks.

2.9.2 Support Vector Machine (SVM)

SVM is a powerful supervised machine learning algorithm that developed by Boser et al., 1992. SVM can be used for both classification and regression tasks (Cortes & Vapnik, 1995). Overall, SVM is a popular and highly effective classifier in machine learning due to its ability to handle complex datasets and produce accurate results (Cervantes et al., 2020). It is particularly effective in solving classification problems, and it has the ability to perform both linear and non-linear classification through the use of the kernel trick to map the data into a higher dimensional space (N-dimensional hyperplane) where the classes can be separated using planes or other functions. In SVM, the choice of kernel depends on the nature of the classification problem. Gaussian and polynomial kernels are commonly employed, with linear kernels being suitable when the number of features exceeds the number of observations. On the other hand, Gaussian kernels are preferred when the number of observations is greater than

the features. Polynomial kernels work well for problems where all the training data is normalized.

The Gaussian kernel, also known as Radial Basis Function (RBF) kernel (this kernel is set with our configuration of the hyperparameters of SVM algorithm), is widely employed in the kernel classification. The RBF kernel is expressed in Equation (2.6) (Wenzel et al., 2017).

$$K(x, y) = \exp[-\gamma \|x - y\|^2] \quad (2.6)$$

Where $\|x - y\|^2$ is the squared norm distance (inner product) between space vectors x and y . γ is a randomization parameter that controls the width of the kernel. For positive values of $\gamma > 0$, smaller distances between vectors will lead to larger RBF kernel values, emphasizing the importance of nearly data points. Conversely, larger distance result in smaller RBF kernel values, reducing the influence of distant data points. The choice of the optimal value for γ is crucial, as it affects the model's ability to accurately classify data.

The polynomial kernel in SVM is used to model the similarity between vectors in a feature space using polynomial functions of the original variables, thereby facilitating the learning of non-linear models. The polynomial kernel is represented in Equation (2.7) (Wenzel et al., 2017).

$$K(x, y) = (\alpha x^T y + c)^d \quad (2.7)$$

Where α is an adjustable parameter, c is the constant term, d is the degree of polynomial, x^T denotes the transpose of x , and y represents another vector in the feature space.

Support Vector Machines (SVMs) have long been recognized for their effectiveness in handling high-dimensional classification tasks, particularly where the decision boundary is well-defined (Cervantes et al., 2020). Their ability to generalize well with a relatively small amount of data makes them an appealing choice for many binary classification problems, including the detection of XSS attacks. In recent studies, such as the work by (H.-C. Chen et al., 2021), SVM has been successfully employed alongside other classifiers like K-Nearest Neighbors (KNN), Logistic Regression (LR), and Random Forest (RF) for real-time prediction of XSS attacks. The SVM classifier was favored in these scenarios due to its speed and lightweight nature, which are crucial for real-time applications.

However, despite its strengths, SVM is not without limitations when applied to the evolving and complex domain of XSS attack detection. Traditional SVM models are static, relying on fixed decision boundaries that may not adapt well to the continuously changing nature of attack patterns (Hosseinzadeh et al., 2021). This rigidity becomes a significant drawback, particularly in environments where data is not stationary, and new attack vectors emerge over time. While SVM provides a strong baseline, its static nature can limit its effectiveness in dynamic scenarios.

To address these challenges, our research integrates the SVM classifier into a more dynamic and adaptive framework—specifically within the DQN-MAFS model, which

supports temporal dynamic feature selection. By embedding the SVM within this reinforcement learning-based system, we leverage its speed and accuracy to assist in evaluating the relevance of selected features, thereby guiding the reinforcement learning agents. In this model, SVM plays a critical role in the reward distribution mechanism among RL agents. As each agent selects features that are most relevant while discarding redundant or irrelevant ones, the SVM classifier evaluates the impact of these actions on classification accuracy. The reward value for each agent is then calculated based on how effectively its actions have contributed to improving accuracy. This approach not only enhances feature selection but also addresses the problem of reward distribution in multi-agent RL systems, particularly in the context of dynamic feature selection—an issue that has been inadequately addressed in existing methods, as discussed in Section 2.7.2 (Table 2.5).

Therefore, while SVM is utilized for its efficiency and classification performance, its integration into the DQN-MAFS model allows us to overcome its limitations by providing a more adaptable and responsive system. This hybrid approach ensures that the model remains effective in detecting new and evolving XSS attack patterns, thus offering a more robust solution compared to traditional SVM-based methods.

2.9.3 Agglomerative Clustering

Agglomerative clustering is a popular clustering algorithm due to its simplicity and ability to handle large datasets (Tokuda et al., 2022). Agglomerative clustering is a hierarchical clustering algorithm (bottom-up) that starts with each data point in its own cluster and then iteratively merges the two closest clusters as one moves up the

hierarchy until a stopping criterion is met. The resulting hierarchy of clusters can be represented as a dendrogram, which is a tree-like diagram that shows how the clusters are nested. At each iteration, agglomerative clustering computes a distance metric between all pairs of clusters and identifies the two closest clusters based on this metric. Several distance metrics are used to measure the distance between pairs of data points, such as the single-linkage, complete-linkage, average-linkage (Unweighted-average linkage or Weighted-linkage), Centroid linkage, Median linkage, or Ward linkage. These metrics differ in how they define the distance between clusters, which can lead to different clustering results. Ward linkage is used in agglomerative clustering to measure the dissimilarity between clusters (Ward, 1963). It specifically minimizes the variance within the newly formed cluster when merging two clusters. The variance is a measure of how spread out the data points are within a cluster, and by minimizing the variance, Ward linkage aims to create compact and well-separated clusters (Tokuda et al., 2022). During the agglomerative clustering process, at each step, the distances between all pairs of clusters are recalculated using the Ward linkage, and the two closest clusters are merged based on these distances until the desired number of clusters is obtained. The pseudocode of agglomerative clustering is presented in algorithm 1 (Schubert, 2020) (J. Huang et al., 2014).

Mathematically, the Ward linkage can be defined as follows:

Given two clusters A and B with sizes $|A|$ and $|B|$, and their respective centroids (mean vectors) μ_A and μ_B , the distance between the two clusters is represented in Equation (2.8) (Schubert, 2020).

$$d(A, B) = \sqrt{\frac{(|A|+|B|)}{(|A|\times|B|)}} \times \|\mu_A - \mu_B\|^2 \quad (2.8)$$

where $\|\mu_A - \mu_B\|^2$ is the squared Euclidean distance between the centroids of clusters A and B .

Algorithm 1: Pseudocode of agglomerative clustering

Input: Data, Distance metric d between any two data points, and Number of desired clusters k Output: A list of k clusters, each containing a subset of the data points Start algorithm 1. Initialize each data point as a separate cluster clusters = [x1, x2, ..., xn] 2. While the number of clusters is greater than k do a. Compute the distance between each pair of clusters using the chosen distance metric Eq. (6) b. Find the pair of closest clusters based on the computed distance: $\text{closest\_pair} = \text{argmin}(\text{distances})$ c. Merge the closest pair of clusters into a new single cluster d. Update the distance matrix to reflect the merge of the closest pair end 3. Return the first k clusters in the list of clusters End algorithm

(Source: J. Huang et al., 2014)

In this thesis, we address the significant challenge posed by the high complexity and heterogeneity of raw textual data, specifically in the context of malicious XSS attack payloads and JavaScript (S. Gupta et al., 2020). This data is typically unstructured, highly variable, and perpetually evolving, which complicates classification tasks due to the use of various evasion strategies, such as obfuscation and encoding techniques (Guo, 2021). The dynamic nature of this data not only introduces substantial variations across different time intervals (temporal drift) but also within the same data chunk (spatial drift), making conventional machine learning (ML) or deep learning (DL) models inadequate in fully capturing and adapting to these complexities.

In this thesis, the agglomerative clustering algorithm is employed to enhance the classification of malicious XSS attacks by organizing raw textual payloads or JavaScript data within data chunks. Agglomerative clustering is adopted to group similar data samples within each data chunk, thereby facilitating a more granular analysis. By clustering the data, we enable the feature selection process to operate not only at the chunk level (temporal) but also at the cluster level (spatial). This dual approach allows us to consider and mitigate the potential drift that may occur both across time intervals and within the samples arriving in the same time interval.

The choice of agglomerative clustering is grounded in its simplicity, robustness, and its proven ability to handle large datasets effectively, as demonstrated by (Tokuda et al., 2022). By leveraging this clustering technique, we ensure that the feature selection process is more adaptive and sensitive to the underlying data structure, leading to more accurate and reliable detection of XSS attacks.

2.9.4 Reinforcement Learning (RL)

The standard RL is framed as an agent's interaction with a Markov Decision Process (MDP) represented by (S, A, γ, P, r) (Kaelbling et al., 1996), as depicted in Figure 2.8. Where S signifies the fully-observed state space of the environment and the agent, A represents the action space. The discount factor γ reflects the importance of intermediate and future rewards in shaping the agent's decisions. The transition dynamics of the environment are described by $P = \{p(s_0), p(s_{t+1}|s_t, a_t)\}$, where $p(s_0)$ denotes the initial state distribution, and $p(s_{t+1}|s_t, a_t)$ represents the state transitions of the environment. Additionally, $r(s, a)$ defines the reward function,

indicating the immediate reward received by the agent when taking action a in state s . At each time step t , the agent, situated in state $s_t \in S$ selects an action $a_t \in A$ based on its policy $\pi(a_t|s_t)$, which such policy represents the modeled map of agent's action selection a_t from the set of available actions A . Subsequently, the agent transitions the environment to state s_{t+1} according to transition dynamics P , and it receives a reward $r_t = (s_t, a_t)$ for the action taken.

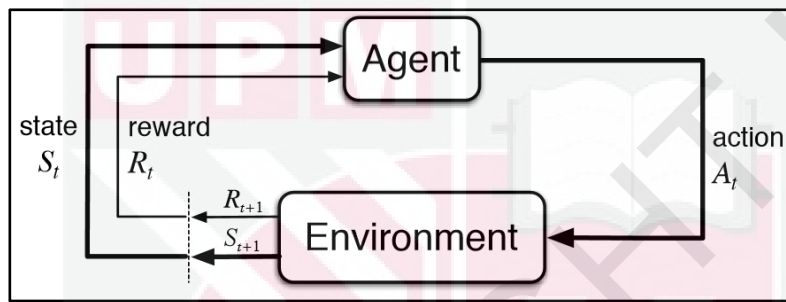


Figure 2.8: Basic structure of Reinforcement learning algorithm
(Source: Kaelbling et al., 1996)

The primary objective of RL is for agent to learn the optimal or nearly-optimal policy: $\pi: A \times S \rightarrow [0,1], \pi(a_t, s_t) = P(a_t|s_t)$, aiming to maximize the γ -discounted cumulative future return, which considers the accumulation of rewards from immediate to future time steps as formulated in Equation (2.9-11) (T. T. Nguyen et al., 2020). The learning process in RL involves exploring and exploiting actions to achieve the best policy for effective decision-making in the given environment.

$$J(\pi) = \mathbb{E}_{s_0 \sim p(s_0), a_0 \sim \pi(a_0|s_0), s_1 \sim p(s_1|a_0, s_0), a_1 \sim \pi(a_1|s_1), \dots, [\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]} \quad (2.9)$$

$$= \mathbb{E}_{s_0, a_0, s_1, \dots \sim p, \pi} [\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)] \quad (2.10)$$

$$\pi^* = \operatorname{argmax}_{\pi} J(\pi) \quad (2.11)$$

Where $J(\pi)$ represents the expected cumulative future return, where the agents interact with environment and selects actions based on $\pi(a_t|s_t)$. The expression $\sum_{t=0}^{\infty} \gamma_t r(s_t, a_t)$ calculates the discounted sum of rewards obtained by the agent over infinite time horizon. The goal is to $J(\pi)$ by finding the policy π^* that yields the highest expected cumulative return considering various state-action sequences.

Alternative approaches involve estimating different functions from the experience data, which can be used to derive the optimal policy (π^*) efficiently. Two common approaches are value-based and model-based methods (Arulkumaran et al., 2017). In value-based method, the focus is on learning the Q-function $Q^\pi(s, a)$, also known as the state-action value function. This function quantifies the future return of taking action a_{t_c} where t_c represents current time from state s_{t_c} and subsequently following policy π as represented in Equation (2.12) (T. T. Nguyen et al., 2020).

$$Q^\pi(s_{t_c}, a_{t_c}) = \mathbb{E}_{s_{t+1}, a_{t+1}, s_t, \dots \sim p, \pi | s_t, a_t} [\sum_{t=t_c}^{\infty} \gamma_{t-t_c} r(s_t, a_t)] \quad (2.12)$$

By estimating the Q-function, it becomes possible to enhance the policy. Specifically, when the Q-function Q^* for the optimal policy π^* is learned, the optimal policy is directly obtained as shown in Equation (2.13) (T. T. Nguyen et al., 2020).

$$Q^\pi(s_t, a_t) = \delta(a_t = \operatorname{argmax}_a(s_t, a)) \quad (2.13)$$

In contrast, model-based methods involve approximating the dynamics $\hat{p}(s_{t+1}|s_t, a_t)$ and the reward function $\hat{r}(s, a)$ by fitting approximators $\hat{p}(s_{t+1}|s_t, a_t)$ and $\hat{r}(s, a)$ from observed transitions (s_t, a_t, s_{t+1}, r_t) . These approximated models are then employed to approximate the optimal policy as formulated in Equation (2.14) (Z. Zhou et al., 2023).

$$\pi^* \approx \operatorname{argmax}_{\pi} \hat{J}(\pi) = \operatorname{argmax}_{\pi} \mathbb{E}_{s_0, a_0, s_1, \dots \sim \hat{p}, \pi} \left[\sum_{t=0}^{\infty} \gamma_t \hat{r}(s_t, a_t) \right] \quad (2.14)$$

In this context, the learned model can also be used to assess the approximation of the state-action value function Q^{π} for any policy π as expressed in Equation (2.15) (Zhou et al., 2023).

$$Q^{\pi}(s_t, a_t) \approx \mathbb{E}_{s_{t+1}, a_{t+1}, s_t, \dots \sim \hat{p}, \pi | s_t, a_t} \left[\sum_{t=t}^{\infty} \gamma_{t-t} \hat{r}(s_t, a_t) \right] \quad (2.15)$$

The quality of approximations to π^* and Q^{π} heavily relies on the accuracy and reliability of the learned dynamics $\hat{p}(s_{t+1} | s_t, a_t)$ and reward function $\hat{r}(s, a)$.

Both value-based and model-based approaches in reinforcement learning involve the trade-off between exploration and exploitation, which is a fundamental concept in RL. The exploration-exploitation trade-off guides the agent's decision-making process in learning the optimal or nearly-optimal policy.

2.9.4.1 Q-learning and Deep Q-Network (DQN)

Q-learning is a foundational reinforcement learning algorithm that forms as the basis for several other RL models, including DQN and double-deep Q-network (DDQN). It was introduced by Chris Watkins in 1989, Q-learning has become widely used due to its effectiveness and versatility. Q-learning is a model-free, value-based algorithm aims to learn an optimal action-value function, denoted as the Q-function (Andrew, 1998), which predicts the expected cumulative reward for taking a specific action in a given state while following a particular policy. The core principle of Q-learning relies

on the Bellman equation for Q^* , a dynamic programming approach as represented in Equation (2.16) (Z. Zhou et al., 2023). By updating the Q-values iteratively, the agent refines its estimates of the Q-function, allowing it to make informed decisions that maximize the expected cumulative reward over time.

$$Q^*(s_t, a_t) = r_t + \gamma \max_a Q^*(s_{t+1}, a) \quad (2.16)$$

Q-learning has been successfully applied to a wide range of problems, including game playing, control systems, and optimization tasks. However, Q-learning faces challenges when dealing with high-dimensional and complex environments, such as those encountered in playing video games directly from pixel inputs (Arulkumaran et al., 2017). In such cases, traditional Q-learning struggles to efficiently represent the state-action space, which hinders its ability to learn optimal policies effectively. To address these limitations and enable reinforcement learning in high-dimensional domains, DeepMind team proposed the Deep Q-Network (DQN) in 2013 (Mnih et al., 2013) (Mnih et al., 2015). DQN combines Q-learning with deep neural networks, allowing it to handle complex state representations more effectively. By employing deep neural networks, DQN can learn abstract and hierarchical representations of the environment and achieved groundbreaking results in learning.

In DQN, each input state is processed through different neural network layers with different weight parameters θ and $\hat{\theta}$. The network generates Q-value outputs corresponding to possible actions. To further enhance learning, DQN employs an experience replay memory, where historical tuples of state, action, reward, and next state (s_t, a_t, r, s_{t+1}) values are stored. The network is trained by sampling small batches from this replay buffer to update the Q-value estimations at each iteration. The

architecture of DRL (Deep Reinforcement Learning) involves effectively seeking and finding the best weight factors from historical data, including Q-values, actions, and state transitions. A key advantage of DQN is that its computational complexity for calculating Q-values and actions remains linear, especially when based on a multilayer perceptron as the underlying neural network. The DQN algorithm utilizes two neural networks – the evaluation and target networks – to learn and approximate the optimal action-value function. The use of experience-replay and a target network aids in overcoming convergence challenges associated with traditional Q-learning in complex environments. By leveraging deep neural networks and experience replay, DQN has demonstrated remarkable success in learning to play video games solely from pixel inputs, showcasing the potential of deep reinforcement learning in handling high-dimensional and challenging tasks.

2.9.4.2 Multi-Agent RL (MARL)

When the realm of sequential decision-making expands to encompass multiple agents, the concept of Markov Game (MG) is commonly employed. The MG was initially presented by Littman (1994) to extend the concept of MDPs to scenarios where multiple agents simultaneously interact within a shared environment, potentially interacting individually or with each other as well. The formalization is designed for a discrete-time setting and is represented as follows (Littman, 1994) (Gronauer & Diepold, 2022).

Definition 1: Markov Game (MG) is an extension of MDP by extending the state of action to set of spaces for actions, the state of reward to set of spaces of rewards. It is

represented by the tuple $(N, S, \{A^i\}, \emptyset, \{\mathfrak{R}^i\}, \gamma)$ where N denotes the number of agents. The environment is affected by joint action space instead of single action $A = A^1 \times A^2 \times \dots \times A^N$. Then these agents receive an associated reward functions that are denoted as $\mathfrak{R}^i: S \times A \times S \rightarrow R$. The actions are selected and executed by each agent based on his own policy $\pi^i: S \rightarrow P(A^i)$. The system evolved from state s_t to s_{t+1} and each agent i should receive a reward R^i .

There are three types of MG based on the designed reward structure that assigned according to agent behaviour that is influenced significantly by type of its own task (Z. Zhou et al., 2023):

1. Fully cooperative stochastic game setting: the reward is distributed equally on the agents $R_1 = R_2 \dots R_N$.
2. Competitive stochastic game setting: the reward is distributed in opposite way $R_1 = -R_2$ and total summation of the rewards equals to zero in any given state transition. This case is occurred when the agents are wary of maximizing their own rewards while minimizing others' rewards.
3. Mixed game: the reward is distributed in mixed way that there are not any restrictions on agent goals. There are two types of mixed games, namely, agent independent and agents aware (joint-action learners).

Definition 2: The value function for MG is defined as the utility in a particular state under joint policy π starting from a particular state $V_\pi: S \rightarrow \mathbb{R}$ and it is calculated based on equation (2.17) (Gronauer & Diepold, 2022). The joint policy is defined $\pi = (\pi^1, \pi^2 \dots \pi^N)$. The goal of the agent is to optimize its own policy π^i given that other agents policies are not fixed. We denote other agents' policies as $\pi^{-i} = \{\pi^1, \dots, \pi^{i-1}, \pi^{i+1}, \dots, \pi^N\}$, Where $1, \dots, -i, i+1, \dots, N$ denote all other agents excluding the agent i .

$$V_{\pi^i, \pi^{-i}}^i(s) = \mathbb{E}_{s_{t+1} \sim \rho, a_t \sim \pi} [\sum_{t=0}^{\infty} \gamma^t R^i(s_t, a_t, s_{t+1}) | s_0 = s] \quad (2.17)$$

Multi-agent RL poses additional challenges compared to single-agent RL due to the increased complexity of the interactions and the presence of other learning agents. Various algorithms have been developed for multi-agent RL, including centralized training with decentralized execution (CTDE), independent learners, and communication-based approaches like differentiable communication.

The challenges in Multi-Agent Reinforcement Learning (MARL), as outlined by (Gronauer & Diepold, 2022; Oroojlooy & Hajinezhad, 2023), encompass non-stationarity due to evolving environments, the complexity of learning communication for effective cooperation, the coordination intricacies arising in multi-agent systems, scalability concerns with a growing number of agents, the partial observability difficulties due to agents may have limited information of current state, and the reward distribution challenge in fully cooperative settings.

In this thesis, our primary objective is to address the dynamic nature of XSS attacks, specifically the temporal feature drift problem, where the relevance of features in attack detection evolves over time. To tackle this, we employ a MARL approach for dynamic feature selection, leveraging RL effectiveness in handling dynamic changes.

Dynamic Feature Selection (DFS) methods have recently emerged as a vital approach in the context of evolving datasets, particularly for addressing the challenge of feature drift. Unlike Static Feature Selection (SFS) methods as explained in Section 2.6, which rely on a predetermined, fixed set of features throughout the entire analysis, DFS methods continuously update the feature set in response to new data. This adaptability

ensures that the model remains effective as the data evolves, making DFS a superior approach in dynamic environments. Reinforcement Learning (RL) algorithms have proven particularly effective in implementing DFS due to their ability to dynamically respond to changes, automate the feature selection process, and learn from the actions taken by agents over time. As a result, RL-based DFS methods often outperform traditional SFS approaches in scenarios requiring sequential decision-making, where adaptability is crucial (Covert et al., 2023).

However, the current state of RL-based DFS methods, as discussed in Section 2.7.2 and Table 2.5, reveals significant challenges and limitations. Existing RL-based DFS methods can be broadly classified into two categories: Single-agent DFS and Multi-agent DFS. Single-agent DFS methods, such as those proposed by (Y. Cheng et al., 2021; Covert et al., 2023; X. Li et al., 2021; Wu et al., 2022; R. Xu et al., 2021), involve a single RL agent responsible for selecting all features. While this design is computationally efficient due to the optimization of a single policy network, it struggles as the feature space expands. The action space increases exponentially with the number of features (K), leading to the potential for suboptimal, locally confined solutions. Conversely, Multi-agent DFS methods, exemplified by works such as (Fan et al., 2020, 2021; Z. Fang et al., 2019; M. Kim et al., 2022; K. Liu et al., 2021; Paniri et al., 2021; Ren et al., 2023), deploy multiple agents, each tasked with selecting specific features. Although this approach better controls the action space, it requires optimizing as many policy networks as there are features, leading to significantly higher computational costs and reduced training efficiency.

A critical gap in current Multi-agent DFS methods is the lack of an effective algorithm for reward distribution, particularly in multi-agent environments. Existing methods do not adequately address the unequal contributions of agents to the overall performance, a problem further exacerbated in parallel multi-agent settings. In such environments, where agents take actions simultaneously, the non-stationary nature of the selected features complicates the accurate allocation of rewards. This results in a challenge where rewards may not accurately reflect each agent's contribution to the selection of relevant features and the exclusion of redundant or irrelevant ones performance (Gronauer & Diepold, 2022; T. T. Nguyen et al., 2020).

To address these challenges, our research proposes a novel adaptive cross-site scripting (XSS) attack detection framework utilizing Multi-agent Reinforcement Learning (MARL) for Dynamic Feature Selection (DFS). This framework, named Deep Q-network Multi-Agent Feature Selection (DQN-MAFS), builds on the foundational concept introduced by Liu et al., 2021, which employs a Multi-agent DFS method by assigning one agent per feature to decide on selection or deselection actions, as detailed in Chapter 4 (Section 4.2). Furthermore, DQN-MAFS introduces a robust sub-model for reward distribution, named Fair Agent Reward Distribution-based Dynamic Feature Selection (FARD-DFS). FARD-DFS includes four distinct reward distribution methods: Accumulative Contribution (ACC), Alternative Contribution (ALC), Impurity-Based (IM), and One Action at One Time (OA-OT), as elaborated in Chapter 4 (Section 4.2.2.5). By enabling simultaneous action changes across agents, DQN-MAFS significantly enhances the exploration of the solution space, improving the efficiency and effectiveness of the learning process, ultimately addressing the identified challenges in the current state of the art.

2.9.5 Deep learning approaches

Deep learning (DL), a subset of machine learning, has gained in prominence due to its ability to comprehend intricate data representations. It finds applications across diverse domains like computer vision, natural language processing (NLP), voice recognition, and anomaly detection (Dong et al., 2019; Geetha & Thilagam, 2021). A standout feature of DL is its automatic feature extraction and abstraction, avoiding the need for manual feature extraction. In the realm of XSS attack detection, DL methods have gained prominence, with various architectures including Convolutional Neural Networks (CNN), Long Short-Term Memory (LSTM), and Bi-directional Long Short-Term Memory (Bi-LSTM) proving especially influential. In this thesis, we adopt the TextCNN model, rooted in the CNN algorithm, optimized for text classification. Additionally, we employ the Bi-LSTM algorithm for word embedding, facilitating dynamic information extraction from raw textual XSS attack samples. The subsequent sub-sections will delve into these specific DL methods background.

2.9.5.1 Convolutional Neural Networks (CNNs)

CNNs are a type of artificial neural network (ANN) specifically designed to extract translation-equivariant responses called feature maps by employing convolutional kernels over input features. A CNN consists of three main layers: convolutional, pooling, and fully connected. In the convolutional layer, different feature maps are computed by sliding convolutional kernels over input data. The process helps to capture local patterns and features within data. Subsequently, the pooling layer reduces the size of convolved features while preserving essential information. Pooling layers

employ various activation functions, such as ReLu, Maxout, tanh and sigmoid, to introduce the nonlinearity required for solving complex, nonlinearly separable problems. Finally, the fully connected layer is responsible for determining association between features and classifying the target classes using an activation function like softmax function (Alzubaidi et al., 2021; H. Xu et al., 2016; Yamashita et al., 2018).

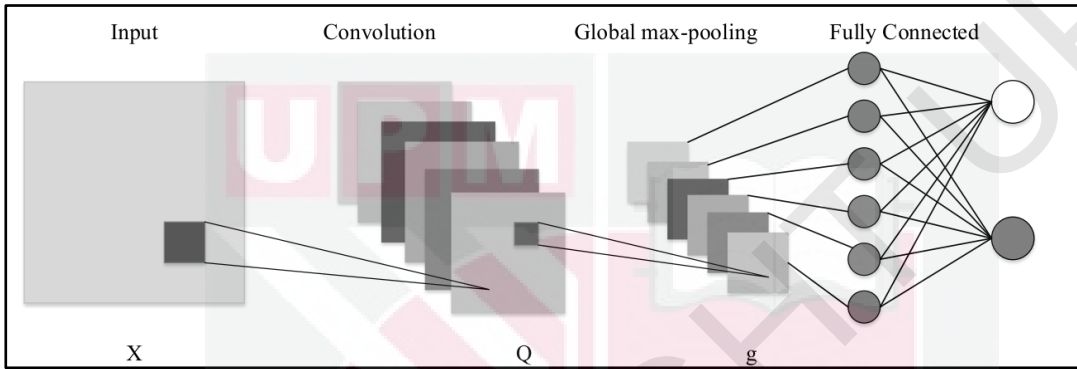


Figure 2.9: Convolution Neural Network architecture
(Source: Alzubaidi et al., 2021)

As shown in Figure 2.9, a typical CNN architecture consists of a convolutional layer, a pooling layer and a fully connected layer. Each convolution layer operates with a several convolution kernels denoted by $k(x)$ with dimensions $(n \times n \times q)$ similar to the dimensions $(m \times m \times r)$ of input sequence $X = (x_1, x_2, x_3, \dots, x_l)$, where $n < m$ and $q \leq r$. The kernels have the same biases b^k and weight w^k for local connections. The convolution layer ($C(y)$) is mathematically represented in Equation (2.18) (Alzubaidi et al., 2021), by calculating a dot product between its input and the weights.

$$h^k = f(w^k \cdot x + b^k) \quad (2.18)$$

For the input sequence $X = (x_1, x_2, x_3, \dots, x_l)$, the convolutional layer employs set of filters F , with the sliding step S to extract local features. For each filter $f_j \in F$, the result q^j in position i as formulated in Equation (2.19) (Alzubaidi et al., 2021).

$$q_i^j = C(f_j^T \cdot \hat{q}_i^{j-1} + b) \quad (2.19)$$

Where $\hat{q}_i^{j-1}$ represents the local region of the input sequences $(x_{i:i+h-1})$, and b is the bias between convolutional layer and pooling layer. Then the output of the convolutional layer will be used as the input of the pooling layer. For all n filters in F , there are n sequences in $Q = \{q^j\}_{j=1}^n$. Hence, it can produce a new sequence with width n and length $l - h + 1$.

The subsequent stage involves down-sampling the feature maps in the sub-sampling layers, which are produced by the preceding convolutional operations. This downsizing reduces the network parameters, leading to faster training and addressing the overfitting problem. Each feature map undergoes a pooling function over adjacent areas of size $p \times p$, where p is the kernel size. This pooling process retains the most significant information or features at each step, effectively summarizing the data. To execute the pooling operation, both the stride and kernel are initially assigned sizes. Various pooling methods are available, but the most common and widely used pooling methods are max pooling, min pooling, and global average pooling. Eventually, the Fully Connected (FC) layers receive the mid- and low-level features and create high-level abstractions, similar to a conventional neural network. The final-stage layers represent the classification scores, achieved through techniques like SVMs or softmax. Each score corresponds to the probability of a specific class for a given input instance.

This process allows the CNN to effectively extract and represent relevant features, leading to accurate and robust classification results. In this thesis, a CNN-based text classification model called TextCNN (Y. Kim, 2014) is used with textual raw XSS data for learning. More details about it are given in the Chapter 4 (Section 4.3.5).

2.9.5.2 Long Short-Term Memory (LSTM)

LSTM, a variant of RNN, is specifically designed to effectively capture and learn long-range temporal dependencies in sequences of varying length (Hochreiter & Schmidhuber, 1997). It was developed to overcome the issue of gradient vanishing that often arises in traditional RNNs. LSTM achieves this by incorporating gate functions, which act as selective filters, allowing only relevant information to pass through while discarding irrelevant data (Gao et al., 2019). The LSTM model is composed of three crucial gates: the input gate, output gate, and forget gate. These gates play significant role in controlling the flow of information within LSTM cell as illustrated in Figure 2.10. In this way, LSTM can retain both long-term and short-term information while efficiently filtering out unwanted data. The forget gate f_t governs the extent to which the previous state c_{t-1} is retained, allowing the LSTM to determine what information should be forgotten from previous cell state. The input gate i_t decides whether the current input should be utilized to update the LSTM information, thereby enabling the model to determine which new information should be incorporated. Lastly, the output gate o_t determines the specific parts of the current cell state that should be outputted to the subsequent layer for further iteration.

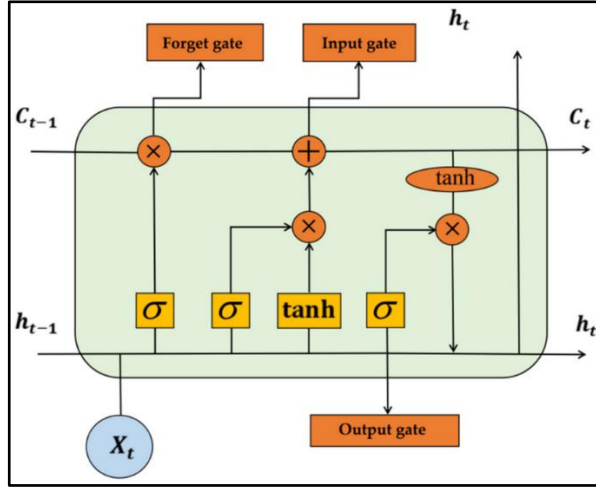


Figure 2.10: Long short-term cell structure
(Source: Hochreiter & Schmidhuber, 1997)

The forget gate within the LSTM cell receives two inputs: the hidden state h_{t-1} from the previous layer's output and the input X_t from the current time step. Through training, the gate function f_t is obtained which calculated using Formula (2.20) (Gao et al., 2019).

$$f_t = \sigma (W_f \cdot [h_{t-1}, X_t] + b_f) \quad (2.20)$$

Similarly, the input gate in the LSTM unit takes the hidden state h_{t-1} and the X_t of the current time step as inputs. The gate function i_t is trained to integrate the information from both the previous and current moments using a small neural network with activation function $\tanh$. The relevant calculations are given by Formulas (2.21) and (2.22) (Gao et al., 2019).

$$i_t = \sigma (W_i \cdot [h_{t-1}, X_t] + b_i) \quad (2.21)$$

$$\tilde{C}_t = \tanh (W_c \cdot [h_{t-1}, X_t] + b_c) \quad (2.22)$$

Following that, the output f_t of the forget gate is multiplied with the global information C_{t-1} from the previous layer to control information selection, allowing for the forgetting of certain parts of the global information. On the other hand, the output i_t of the input gate function is multiplied with integrated information $\tilde{C}_t$ to selectively incorporate new information. This is expressed by Formula (2.23) (Gao et al., 2019).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (2.23)$$

In the LSTM cell, the output gate takes the hidden state h_{t-1} and the input x_t of the current time step as inputs. The gate function o_t is trained to regulate the output of the current moment, determining how the global information should impact the next LSTM cell. The calculation for the output gate is given by Formula (2.24) (Gao et al., 2019).

$$o_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o) \quad (2.24)$$

The global information c_t of the current time step is then multiplied with output o_t of the output gate function by a $\tanh$ activation function to obtain the hidden state h_t of the current moment, where the global information is considered in the next LSTM cell. This calculation is represented by Formula (2.25) (Gao et al., 2019).

$$h_t = o_t * \tanh(C_t) \quad (2.25)$$

In the aforementioned formulas, W_f , W_i , and W_o represent the weight of matrices for forget gate, input gate, and output gate, respectively. b_f , b_i , b_o , and b_c are bias vectors. The symbol $*$ represents an element-wise (element-by-element) multiplication, and σ

represents the activation function which is called sigmoid. $\tilde{C}_t$ represents the state of the memory cell at time t .

LSTM has gained popularity as a variant of RNN due to its ability to effectively handle time-series data for classification and prediction tasks. Numerous studies have utilized LSTM for various applications, including XSS detection, as observed in studies by (Y. Fang, Li, et al., 2018; Lei et al., 2020; Stiawan et al., 2023). Additionally, a study proposed an integrated detection method using LSTM algorithm and CNN model (R. Yan et al., 2018). One significant issue is that the training process of LSTM networks is inherently sequential and not easily parallelizable. This lack of parallelization results in resource-intensive implementations that demand substantial computational power and time, especially when dealing with large datasets or real-time detection scenarios. This limitation makes it difficult to scale LSTM models for extensive or high-dimensional data, which is often the case with XSS detection tasks where the data is heterogeneous and voluminous (Bai et al., 2018).

2.9.5.3 Bi-directional Long Short-Term Memory (Bi-LSTM)

Bi-LSTM is a variation of the LSTM algorithm that consists of two LSTM networks (Graves & Schmidhuber, 2005). These LSTM networks process input in two opposite directions: forward, and backward. Unlike traditional LSTM, which only allows input to flow in one direction, Bi-LSTM enables bidirectional flow, preserving both future and past information. This bidirectional feature empowers the network to establish context for each character in the input text, considering both its preceding and succeeding context. By incorporating information from both directions, Bi-LSTM achieves high detection accuracy in various applications.

In contrast to One-way LSTM, which capture sequence information only in a forward-to-backward order, Bi-LSTM comprehensively considers contextual semantic relationships and learns script features by utilizing both forward and backward propagation as depicted in Figure 2.11. This enables Bi-LSTM to extract features that are highly relevant to script classification. The hidden layer state H of the Bi-LSTM is obtained by concatenating the hidden layer state component $\vec{h}_t$ obtained through forwarding propagation with hidden layer state component $\overleftarrow{h}_t$ obtained by backward propagation. This way, Bi-LSTM capitalize on bidirectional information flow to enhance its understanding of the input data and improve its performance in various tasks.

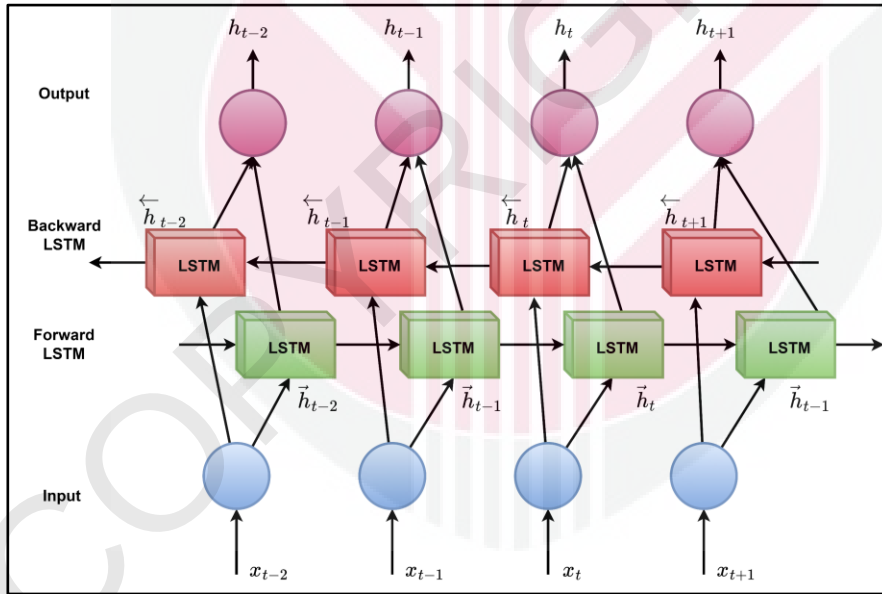


Figure 2.11: Structure of the Bi-LSTM network
(Source: Z. Huang et al., 2015)

In domain of the XSS attack detection, Bi-LSTM was integrated with the CNN algorithm to propose a model for XSS attack detection in the study conducted by (Hu et al., 2023). (J. Yang et al., 2020) use a single layer of Bi-LSTM with the Word2Vec

model. Bi-LSTM was used to generate encoded features and learn the semantics from the word level and segment level. Also, in the study of (Y. Huang et al., 2021), two layers of Bi-LSTM with the Word2Vec model are presented to extract more information from the hidden states. (Y. Fang, Xu, et al., 2020) also combined the Word2Vec model with a single Bi-directional RNN (Bi-RNN) layer.

We found that the current studies use only single layer of Bi-LSTM layer for capturing long term dependencies in both directions. However, because of the high heterogeneity and complexity of the textual data, this Bi-LSTM is still not enough. Because the Textual data complexity and evolving nature of XSS attack, the features are non-stationary and caused polesemy issue and especially with JavaScript-based data, thus it is required more layers of Bi-LSTM layers to extract and represent the feature in more effective way that resulted into detecting the evolved attacks efficiently. In this thesis, we use two layers of Bi-LSTM algorithm in combination with word embedding method as a dynamic word embedding method for feature learning purposes to extract the complex and hidden information from the raw textual XSS data.

In this thesis, we use two layers of Bi-LSTM algorithm in combination with word embedding method as a dynamic word embedding method for feature learning purposes to extract the complex and hidden information from the raw textual XSS data.

In the domain of XSS attack detection, Bi-LSTM (Bidirectional Long Short-Term Memory) networks have been employed to enhance the understanding of contextual relationships within textual data. For instance, Hu et al. (2023) integrated Bi-LSTM with CNN to propose a model that captures both sequential dependencies and spatial features of input data. Similarly, Yang et al. (2020) utilized a single layer of Bi-LSTM

combined with the Word2Vec model to generate encoded features, aiming to capture semantics at the word and segment levels. Huang et al. (2021) advanced this approach by employing two layers of Bi-LSTM alongside Word2Vec to extract more comprehensive information from the hidden states, enhancing the richness of feature representation. Fang et al. (2020) also leveraged Bi-LSTM in conjunction with Word2Vec, albeit using a single Bi-directional RNN layer.

Despite these advancements, the current state of the art primarily relies on a single layer of Bi-LSTM to capture long-term dependencies in both directions. However, given the high heterogeneity and complexity of textual data, especially in the context of XSS attacks, a single Bi-LSTM layer may be insufficient. XSS attacks often involve complex, evolving JavaScript-based payloads that introduce non-stationary features and polysemy, where words or symbols have multiple meanings depending on the context. This complexity requires more advanced methods to effectively capture and represent the intricacies of the data.

To address these challenges, this thesis proposes the use of two layers of Bi-LSTM in combination with a dynamic word embedding method for feature learning. The additional Bi-LSTM layer allows for a deeper and more nuanced extraction of hidden information from raw textual XSS data, thereby improving the model's ability to detect evolved attacks. By stacking multiple Bi-LSTM layers, we extend the model's capacity to capture subtle and complex patterns within the data, providing a more robust representation that is better suited for the dynamic nature of XSS attacks. This approach not only enhances feature extraction but also improves the overall effectiveness of the model in detecting sophisticated and evolving threats.

2.9.6 Tabu Search Overview

The tabu search, proposed by Glover (Fred Glover, 1989; Glover & Laguna, 1998), is a meta heuristic method that can be used in a form of local neighbourhood search to solve combinatorial optimization problem. It has received widespread attention recently. Its flexible control framework and several spectacular successes in solving NP-hard problems caused rapid growth in its application. It differs from the local search technique in the sense that tabu search allows moving to a new solution which makes the objective function worse in the hope that it will not trap in local optimal solutions. Tabu search uses a short-term memory, called tabu list (TL), to record and guide the process of the search. In addition to the tabu list, we can also use a long-term memories and other prior information about the solutions to improve the intensification and/or diversification of the search.

The tabu search scheme can be outlined as follows: Each feasible solution, $S \in \Omega$ has an associated set of neighbors, $N(S) \in \Omega$ where Ω is a set of feasible solutions. A solution $S^* \in N(S)$ can be reached from current solution S by an operation called a "move" to best solution S^* . Tabu search moves from a solution to its best admissible neighbor, even if this causes the objective or fitness function to deteriorate. This is non-greedy behaviour of tabu search which is also helpful to avoid trapping in local optima.

To avoid cycling and trapping into local minima, solutions that were recently explored are declared as "forbidden" or "tabu". These recently visited solutions are kept in a list which is known as tabu list (TL) for a number of iterations (Q). For every move, (TL)

is consulted each time. If a move is present in the TL , then algorithm discards that move and forwards to next iteration. The tabu status of a particular solution can be overridden when aspiration criteria or level (AL) is satisfied. Sometimes intensification and diversification strategies are used to improve the search. In the first case, the search is accentuated in promising regions of the feasible domain. In the second case, an attempt is made to consider solutions in a broad area of the search space.

Algorithm 2: pseudocode of Tabu search algorithm used for dynamic feature selection

Parameters:

Ω : Set of feasible solutions

S : Current solution

S^* : Best admissible solution

$f(cost)$: Objective function

$N(S)$: Neighborhood of S

V^* : Sample of neighborhood solution

TL : Tabu List

AL : Aspiration Level

Q : Fixed number of iterations

Start with initial feasible solution $S \in \Omega$

Initialize TL and AL

for Q **do**

 Generate neighbor solution $V^* \subset N(S)$

 Calculate fitness $Cost(V^*)$ and find best $S^* \in V^*$

if neighbor S^* from S is not in TL **then**

 Accept the neighbor and update the current solution S

 Update TL and AL

 Increment iteration number

else

if $cost(S^*) < AL$ **then**

 Accept the neighbor and update current solution S

 Update TL and AL

 Increment iteration number

In response to the complexity of spatial and temporal dynamic feature selection, the Tabu Search algorithm emerges as a robust solution. This algorithm empowers a local search mechanism that commences from an initial solution and systematically transforms it through various operations. If an operation results in an improved

solution, it is accepted; otherwise, the process continues iteratively until an optimal solution is attained. This iterative approach is vital for addressing the evolving nature of data streams where feature relevance can shift over time.

2.10 Chapter summary

In this chapter, a comprehensive literature survey was conducted, beginning with an overview of XSS attacks, including their types, risk scenarios, and evolution strategies employed by attackers to generate new variations of XSS attacks. The chapter also introduced classical XSS attack detection approaches and its challenges. Subsequently, it provided insight into ML and DL algorithms and their adoption for XSS attack detection. Additionally, it discussed challenges in classifying data streams, including concept drift and its special form, feature drift. Furthermore, the chapter conducted an extensive review of preprocessing methods, encompassing data cleaning, feature extraction, and feature selection. This review highlighted the prevalent focus on these methods for enhancing performance. The subsequent analysis explored related works that employed ML and DL methods for XSS attack detection, revealing their predominant reliance on offline learning and testing strategies, statistical-based feature extraction methods, static word embedding, and classical feature selection methods often overlooking the dynamic nature of data and continuous knowledge updates. Then this chapter centered on RL-based dynamic feature selection, examining the role of reinforcement learning in adapting to feature drift. Finally, this chapter identified the open challenges and gaps in the XSS attack detection domain that serves as the foundation for Chapter 3, which presents the detailed of the proposed methodology.

CHAPTER 3

RESEARCH METHODOLOGY

3.1 Introduction

Cross-site scripting is the most frequent cyber security attack in the web application domain. Compromising such an attack resulting in grievous harm to victims including gaining control over login cookies, passwords, and authentication tokens, and performing application-level actions, whenever the browser renders and executes the vulnerable web page script code. The traditional detection methods such as static, dynamic and hybrid analysis could not detect the unknown forms of XSS attack. To tackle such issues, traditional machine and deep learning techniques were adopted in previous works. However, aforementioned techniques are limited to detect the continuously evolving nature of XSS attacks because of its limitation to dynamic awareness equipping to detect the feature drift that may be caused by the attack evolution. Thus, this thesis aims to propose a framework with temporal and spatial feature drift awareness ability to detect the evolved XSS attacks that helps to overcome the issues of this research. The methodology of the proposed framework is presented in this chapter. This chapter is organized as follows: section 3.2 presents the methodology design of the proposed framework. Section 3.3 defines the problem formulation. Section 3.4 presents the overview of the proposed DA-XX-AD framework.

3.2 Methodology Design

The methodology of designing dynamic and adaptive cross-site scripting (XSS) attacks detection (DA-XSS-AD) unified framework and accomplishing both multi-agent deep Q-network and tabu search-based tempo-spatial dynamic algorithm is presented in Figure 3.1. As it is presented, the methodology starts with a phase of a thorough literature survey that consists of three parts: the first one is a survey about deep learning/machine learning based XSS detection. The second one is a survey about the deep learning based XSS detection. The third one focuses on dynamic feature selection handling of feature drift. The outcome of this phase is three elements, namely, problem statement, scope, and both benchmarks approach and dataset selection. Afterwards, the second phase is dedicated for accomplishing the DA-XSS-AD unified framework for XSS attack detection. This framework is distinguished from existing ones in the literature by its capability of handling various XSS attack forms and scenarios by considering different types of data representation for XSS, namely, JavaScript, payloads, and numerical data. Current literature (Chapter 2 - Section 2.7) often limit their scope to a single data type, which may overlook critical patterns present in other forms of data. By contrast, our framework ensures a more comprehensive detection capability by integrating both data types, thereby enhancing the robustness and adaptability of the detection process. Such capability is directly tied to the gaps identified in the literature review, where it was noted that existing solutions do not fully exploit the potential of different data types for XSS detection. The DA-XSS-DFS framework, therefore, handles an evolution in XSS detection methodologies, addressing this critical need.

The second phase contains two steps, namely, components selection for the framework, and component integration. The outcome of this phase is a unified framework that has two attributes, namely, model capacity by enabling more than one XSS representation data for detection, and evolving awareness by enabling dynamic feature selection. Considering that we have two types of data representation for XSS: numerical and textual. The third phase is dedicated for accomplishing XSS detection from the first type which is numerical and the fourth phase is dedicated for accomplishing detection from the second type which is textual. The outcome of the third phase is a DQN-based multi-agent features selection (DQN-MAFS) algorithm. In order to accomplish the third phase, we adopt the model of RL which is capable of dealing with the temporal dynamic feature selection while we adopt for the fourth phase the framework of Tabu search which is capable of dealing with both spatial and temporal dynamic.

Hence, for the third phase, we perform several steps, namely, data type determination, model design which will be multi-agent model due to the high dimension of features, defining the elements of RL that includes state, action, and reward. The reward schema will handle the issue of rewarding distribution due the existence of multi-agents aiming to accomplish the same goal. The fourth phase will also include data type representation, model design, configuration of Tabu search which includes decision space and objective space definition. The outcome of this phase is the Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS) model. The outcomes of Phases 3 and 4 are integrated within the framework to provide a unified detection model capable of simultaneously processing and evaluating both data types. This dual-path approach is essential because numerical and textual data often present different

characteristics and require distinct analysis techniques. By separating the detection processes while ensuring their results converge, the framework maximizes detection accuracy and reduces the likelihood of false positives or negatives.

The last phase is the evaluation phase, that includes several components, namely, the evaluated algorithms both our developed ones and the benchmarks, and the evaluation metrics. The outcome of this phase is comparison and analysis.



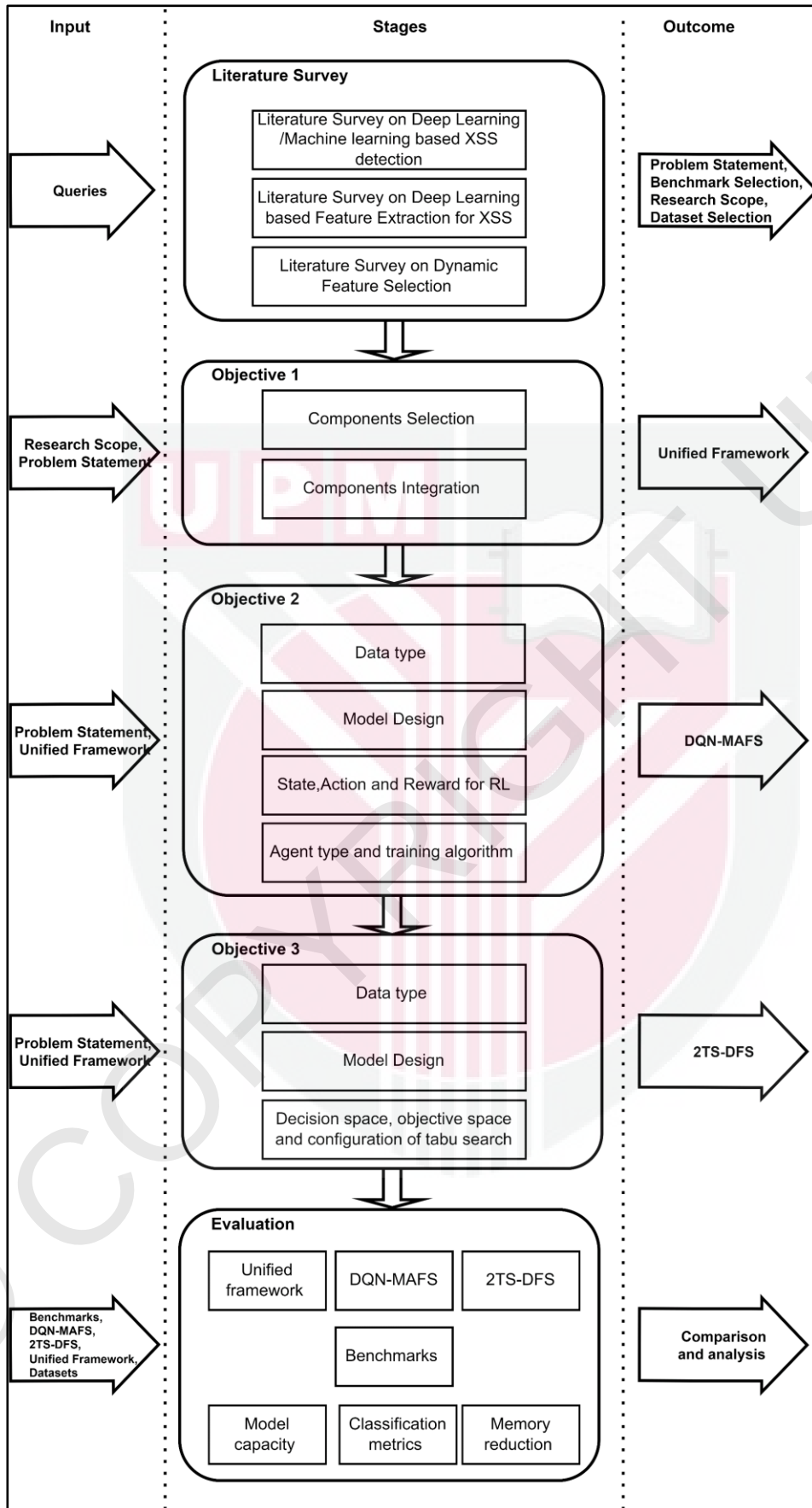


Figure 3.1: Methodology design overview

3.3 Problem Formulation

Assuming a stream data set $D = (x_t, y_t)$ with partial labelling where:

t denotes the time and x_t denotes the feature vector $x_t \in \mathbb{R}^m$

$$y_t = \begin{cases} y_{target,t}, & \text{if label is available} \\ \nabla, & \text{if label is not available} \end{cases} \quad (3.1)$$

The percentage of labelling is $R_t \in [r_{min}, r_{max}]$ indicates the number of labelled samples over the total number of samples within a small time-interval T .

The goal is to build a prediction of the non-labelled samples of the data stream $f(x_t) = \tilde{y}_t$ where the error of the prediction is minimized or $e = \sum_{t=t_0}^{t_c} (\tilde{y}_t - y_{target,t})$ is minimum where t_0 denotes the starting time and t_c denotes the current time.

The function f contains feature selector function $g(.)$. Its mission is to assign for each feature $i = 1, \dots, m$ a binary value $b_i \in \{0,1\}$ where b_i denotes the decision of whether the feature is selected or not. This process is to minimize the error e .

Assuming that after pre-processing, all x_t that comes in the same moment t belong to $X_t \in \mathbb{R}^{m_t \times n}$ where m_t denotes the number of records in chunk X_t , n denotes the maximum number of features in chunk X_t . The goal is to build a mathematical function $g(.)$ where $g(X_t) = \{x'_{1,t}, x'_{2,t}, \dots, x'_{i,t} \dots x'_{m,t}\}$ and $x_{i,t} \in \mathbb{R}^{1 \times n_i}$. The function $g(.)$ can be represented by $g(X_t) = X_t \otimes M_t$ where $M_t \in \{0,1\}^{m_t \times n}$, $\otimes$ denotes element wise product.

Hence, we need to find M_t that maximizes the testing accuracy over the non-labelled samples by leveraging the labeled samples. It is important to note that in the special case when $\forall a_{i,k}, a_{jk} \in M_t$ then $a_{i,k} = a_{jk}$, that means the feature selection is only temporal (or chunk level). Otherwise, when this constraint is violated then the feature selection is spatial and temporal (or instance level).

The function $g(.)$ will be formulated differently according to the nature of data that is represented by x_t which it has two types of representation, namely, JavaScript or payloads as explained in Chapter 2 (Section 2.2.4). Also, it will be built according to the type of features extracted from payloads, numerical or word embedding features. In the case of numerical features, the mission of $g(.)$ will be simplified to temporal feature selection while in the case of words embedding features, the mission of $g(.)$ includes both temporal and spatial feature selection at the same time.

Mathematically, we write the problem as an optimization problem according to the following formula (3.2):

$$M_t^* = \underset{e}{\operatorname{argmin}} = \underset{e}{\operatorname{argmin}}(\sum_{t=t_0}^{t_c} (\widetilde{y}_t - y_{\text{target},t})^2) = \underset{e}{\operatorname{argmin}}(\sum_{t=t_0}^{t_c} (f(x_t) - y_{\text{target},t})^2) \quad (3.2)$$

3.4 DA-XSS-AD Framework Overview

The proposed Dynamic and Adaptive Cross-Site Scripting (XSS) Attacks Detection (DA-XSS-AD) framework for feature drift-aware XSS attack detection exhibits high model capacity as it handles different data representations through two distinct operational modes, as depicted in Figure 3.2. The first mode, DQN-MAFS (DQN-

based Multi-Agent Feature Selection), focuses exclusively on payload data with numerical feature extraction. The second mode, called 2TS-DFS (Tabu Search based Tempo-spatial Dynamic Feature Selection), is designed for JavaScript (JS) or payload data and utilizes text-based processing techniques.

The first DQN-MAFS mode, the framework exclusively deals with numerical features extracted from the payload data. It employs a multi-agent-based reinforcement learning approach to perform temporal dynamic feature selection. The selected features are subsequently fed into a set of machine learning classifiers to discern malicious XSS attacks from benign samples. The detail of multi-agent is presented through the elements of RL design, namely, state, action, and reward plus reward distribution for handling the stochastic game aspect in order to enable convergence as explained in Chapter 4 (Section 4.2). On the other hand, in the 2TS-DFS mode, the framework incorporates a pre-processing phase that includes adaptable context analysis. This analysis involves static word embedding and two layers of Bi-LSTM-based dynamic word embedding. The resulting features are combined using a weighted sum, and the selected features are then fed into a TextCNN-based classifier for XSS attack detection.

Regardless of the mode, the flow of data within the framework remains consistent. Input data, presented in the form of chunks with varying timestamps, undergoes pre-processing to extract statistical or hidden state features. These features are then used in the respective operational mode for feature selection and classification. The framework operates with distinct, separated detection modes—one for raw textual data and another for numerical data (keeping in mind that the numerical data is already

processed data from textual data). For the benchmarking purpose, we used different datasets, two for textual data (JavaScript dataset and payload dataset) and four for numerical payload keeping in mind that only one detection mode is activated at a time according to input data. This separation ensures that there is no overlap or conflict between the detection modes, allowing each to operate independently and optimally.

By providing distinct modes for processing two types of data, the framework ensures comprehensive and effective XSS attack detection. Leveraging the strengths of both 2TS-DFS and DQN-MAFS, it adapts to the specific characteristics of each data type and employs most appropriate techniques for each mode. This approach not only enhances the framework's flexibility but also ensures that the data flow does not impact the integrity or functionality of the detection processes. Ultimately, this leads to improved performance in detecting evolving XSS attacks, as each mode is tailored to handle its specific data representation effectively.

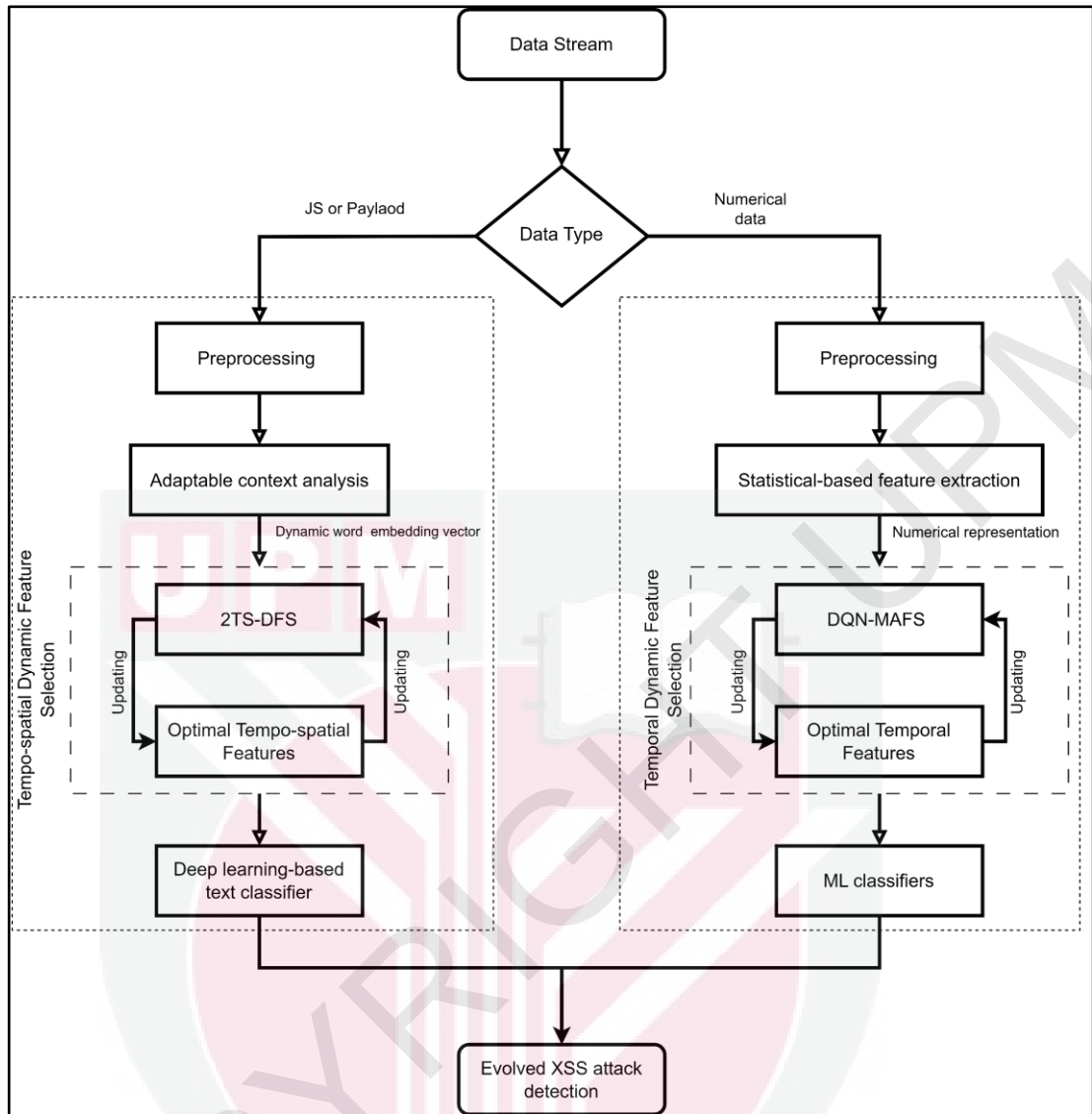


Figure 3.2: DA-XSS-AD framework

3.5 Considerations and Conceptual Basis of the DA-XSS-AD Framework

The design of the DA-XSS-AD framework is rooted in a thorough analysis of the challenges and requirements associated with detecting XSS attacks, particularly in the context of evolving attack patterns and dynamic data environments. Several critical elements and conceptual models were considered in the development of this framework, each addressing specific limitations observed in existing models and methodologies.

1. Addressing Feature Drift and Evolving Attack Patterns:

One of the primary motivations behind the DA-XSS-AD framework was the need to effectively manage the phenomenon of feature drift, which occurs as the characteristics of XSS attacks evolve over time. Traditional static feature selection (SFS) methods, which rely on a predetermined set of features for the entire dataset, are inadequate for handling such changes. As the relevance of features can shift, a dynamic feature selection (DFS) approach was essential to maintain high detection accuracy. The DA-XSS-AD framework is designed to continuously update and refine its feature set as new data emerges, ensuring that the detection model remains responsive to the latest attack vectors.

2. Integration of Incremental Learning:

The framework incorporates incremental learning techniques to enhance its ability to adapt to new data without requiring complete retraining. This is particularly important in real-world applications where data is continuously generated, and the model must update its knowledge base in an efficient and resource-conserving manner. Incremental learning allows the DA-XSS-AD framework to handle streaming data, thereby improving its applicability in dynamic environments where the threat landscape is constantly evolving.

3. Separation of Detection Modes:

A key architectural decision in the framework was the separation of detection modes for different types of data, specifically Textual data (e.g., JavaScript and payload data) and numerical data (preprocessed/secondary data). Each detection mode operates

independently, ensuring that only one mode is active for a given data type at any time. This design choice eliminates potential conflicts between different data processing pipelines and allows the framework to tailor its detection strategies to the unique characteristics of each data type. This separation ensures that the flow of data within the framework does not interfere with the distinct processes required for effective XSS detection.

4. Application of Multi-Agent Reinforcement Learning (RL):

The framework leverages multi-agent reinforcement learning (RL) for dynamic feature selection, a methodology that has shown promise in other domains for managing complex, sequential decision-making processes. Specifically, the DA-XSS-AD framework builds on concepts from existing multi-agent RL systems, such as those discussed by Liu et al. (2021) but introduces a more refined reward distribution mechanism. This mechanism, termed fair agent reward distribution (FARD-DFS), addresses a significant gap in the literature: the need for a reward schema that accurately reflects the contributions of individual agents (or features) to the overall performance of the detection system. By enabling simultaneous action changes among agents, the framework increases the efficiency of the learning process and accelerates the exploration of the solution space.

5. Adaptability and Modularity:

Finally, the framework was designed with adaptability and modularity in mind. It is structured to accommodate a wide range of datasets and XSS attack types, making it versatile in its application. The modular design allows for easy updates and the incorporation of new techniques as they become relevant, ensuring that the framework remains at the cutting edge of XSS detection research.

6. Dynamic Feature Selection:

The use of DFS is a direct response to the limitations of SFS methods, which fail to adapt to changing data characteristics as explained in Chapter 2 (Section 2.6). By dynamically adjusting the feature set, the DA-XSS-AD framework ensures that it remains effective in the face of evolving attack patterns, a capability that is crucial for real-time cybersecurity applications.

7. Existing XSS Detection Models:

While the framework draws on insights from previous XSS detection models, particularly those employing deep learning and NLP techniques, it differentiates itself by incorporating temporal and spatial dynamic feature selection as well as incremental learning capabilities. This allows the DA-XSS-AD framework to address the shortcomings of static models and enhance its ability to detect novel and evolving XSS attacks.

CHAPTER 4

IMPLEMENTATION OF DA-XSS-AD'S MODES

4.1 Introduction

In the previous chapter, the research methodology of the proposed DA-XSS-AD framework was presented. The DA-XSS-AD framework consists of two distinct operational modes. The first mode, DQN-MAFS (DQN-based Multi-Agent Feature Selection), supports temporal dynamic feature selection and focuses exclusively on payload data with numerical feature extraction. The second mode, called 2TS-DFS (Tabu Search based Tempo-spatial Dynamic Feature Selection), provides dynamic feature selection in both temporal and spatial dimensions and is designed for textual raw data, including JavaScript (JS) and payload data. The methodology of the DQN-MAFS and 2TS-DFS models are presented in this chapter. This chapter is organized as follows: Section 4.2 identifies components of the DQN-MAFS model of the suggested framework. This section also presents the deep details of each sub-components of the DQN-MAFS model into sub-sections. While the 2TS-DFS model and its sub-components of the proposed framework is introduced in section 4.3. Section 4.4 states a brief description of the used datasets, the performance metrics used in this thesis to evaluate both proposed modules, and the experimental design employed for configuring the proposed algorithms and models. Finally, a summary of the proposed framework and the methods adopted in this thesis is given in section 4.5.

4.2 DQN-based Multi-Agent Features Selection (DQN-MAFS)

As it has been presented in the problem formulation in Chapter 3 (Section 3.3), there are two types of data representation, namely, JavaScript and payloads as illustrated in Chapter 2 (Section 2.3.3). In the case of numerical data, there is a mode of dealing with numerical features that extracted from the payload using statistical-based feature extraction method. In such a mode, we enable the DQN-MAFS model for performing temporal dynamic feature selection. As shown in Figure 4.1, DQN-MAFS model consists of three main stages: (1) preparation of data chunking which is responsible for generating stream of data chunks from original XSS data (Section 4.2.1). (2) DQN-MAFS framework, which is responsible for dynamic feature selection (Section 4.2.2). (3) Detector which is responsible on classifying the records according to their feature's values (Section 4.2.4).

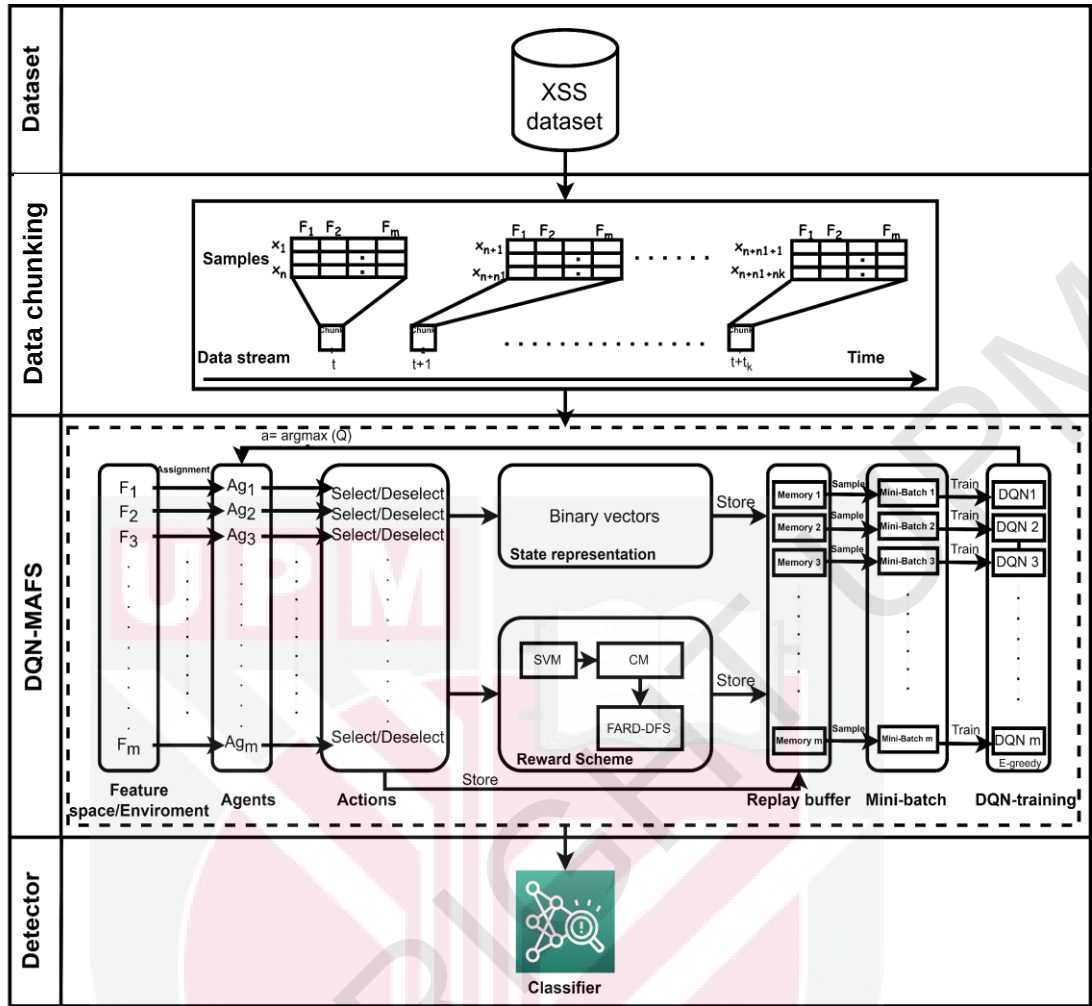


Figure 4.1: The overall DQN-MAFS model

4.2.1 Preparation of Data Chunking

The literature contains a wide range of use of the XSS data with ML/DL approaches as presented in Chapter 2 (Section 2.7.1). However, they all assume batch learning, and no sequential chunks were generated as highlighted in Chapter 2 (Section 2.8). In order to enable the incremental learning approach, we decompose the data into chunks as we provide them to the framework. The chunks are assumed based on the pre-given number of chunks. The time has not been used as a feature, but it helps in ordering the records (sequence order of the data). The data is presented as a stream of chunks to the

method based on Equation (3.1). Each chunk consists of a certain number of rows representing the records $x_1, x_2, x_3, \dots, x_n$ and certain number of columns representing the features $f_1, f_2, f_3, \dots, f_m$. The dimension of features is fixed, while the number of rows changes from one chunk to another according to the way the data is generated, as depicted in Figure 4.1.

Furthermore, the chunk decomposition is not an inherent part of the developed algorithm; rather, it is a component of the experimental design, as detailed in Section (4.4.4). The number and size of chunks are determined based on the dataset's characteristics and the objectives of the experiment. Specifically, larger datasets are divided into larger chunks, under the assumption that no significant feature drift occurs within a single chunk, allowing the focus to remain on detecting drift between chunks. Therefore, the chunk size was empirically chosen to effectively capture drift while maintaining manageable computational costs. This chunking strategy is crucial for structuring the data to align with the goals of dynamic feature selection and drift detection, ensuring that the framework adapts effectively to evolving data patterns.

4.2.2 DQN-MAFS Model

The model of DQN-MAFS is presented in Figure 4.1. The model supports multi-agent learning for feature selection, like (K. Liu et al., 2021) in general and different in details of various components such as state representation, reward schema, exploration, and exploitation policy. In our suggested DQN-MAFS, each agent is responsible for one feature, and it has one of two actions, namely, selection or de-selection, as described in Subsection 4.2.2.4. Each agent generates states from the data

chunks (see Subsection 4.2.2.2), which are then used to determine whether to select or de-select the corresponding feature. The aim is to reduce the data to a more discriminative subset that reflects most relevant feature and improves prediction and efficiency. The agent contains a deep Q-network (DQN) trained based on a mini-batches sampled from an experience-replay buffer, similar to the approach described by (Mnih et al., 2013), as detailed in Subsection 4.2.2.6. The DQN consists of a neural network that is trained using a loss function optimized through gradient descent. The linkage and the overall operation of the DQN-MAFS, including the training process, are thoroughly detailed in Section 4.2.3, which presents the general algorithm encompassing all components of the DQN-MAFS. The following subsections provide the details of the elements of our proposed DQN-MAFS including environment, state, agent, action, and reward schema.

4.2.2.1 Environment

In general, the environment represents all entities outside the agent. In this research, the environment represents the data in terms of the number of features, their types, values, and their associated labels (ground truth) that provides the capability of learning the knowledge based on the calculated reward.

4.2.2.2 State

The state is represented by a binary vector that has a length equal to the number of features. The vector elements are zeros or ones based on the last generated decision of the agents for selecting or deselecting their features. The selection is encoded with one,

while the de-selection is encoded by zero. Hence, the model maintains Markov Decision Process (MDP) or

$$p(S_t|S_{t-1}, S_{t-2}) = p(S_t|S_{t-1}) \quad (4.1)$$

4.2.2.3 Action

The action is associated with the agent's decision of selecting or deselecting its feature. Assuming that we have a total m features, then the selection decision is a binary value ranging from 1, which is the decision to select the first features only to 2^m , which represents the decision of selecting the entire set of features. This is given by the Equation (4.2).

$$a_t = \begin{cases} 1, & \text{when the feature is selected} \\ 0, & \text{when the feature is deselected} \end{cases} \quad (4.2)$$

$a \subseteq A$ where $|A| = 2^m - 1$ and $a = (a_1 \ a_2 \ \dots \ a_t)$

4.2.2.4 Agent

The modelling of feature selection will be based on multi-agent reinforcement learning. Each reinforcement learning agent i^{th} is responsible for making a decision of selecting a subset of features based on the state. The agent is supposed to capture the knowledge generated from the subsequent instances of records, states $s_{i,t}$, action $a_{i,t}$, next state $s_{i,t+1}$, and the reward $r_{i,t}$ obtained by the taken action. To facilitate the agent's learning, a replay buffer is utilized to store instances in the format $(s_{i,t}, a_{i,t}, s_{i,t+1}, r_{i,t})$, which are generated as the agents take different actions over time.

Subsequently, as described in Section 4.2.2.6, these stored instances are sampled from the replay buffer to train each agent's DQN using a batch learning method. Training the DQN in this manner allows the agent to learn from a diverse set of past experiences, which enhances the agent's ability to make more informed and optimal decisions in future steps.

4.2.2.5 Reward Scheme

The role of reward distribution is to assign to each agent a reward value equivalent to its contribution to changing the overall prediction accuracy. Considering that the agents are operating simultaneously, it is difficult to separate the contribution of each agent. Consequently, a non-stationary environment will occur. To resolve this, we propose different methods for reward distribution. Each of these methods is based on changing the decision within a certain number of iterations and calculating the corresponding accuracy that has resulted from the overall decisions of all agents. We assume that the set of combinations of the various decisions is $a = (a_{i,1} \ a_{i,2} \ \dots \ a_{i,3} \ \dots \ a_{i,k})^T$ where each a denotes the joint action of all agents. In addition, we assume that the corresponding accuracy of each decision is given as $Ac = (ac_1 \ ac_2 \ \dots \ ac_i \ \dots \ ac_k)^T$ where k denotes the number of combinations. The reward for each agent is derived from one of the four suggested mathematical models: Accumulative Contribution (ACC), Alternative Contribution (ALC), Impurity (IM), and One-Action at One-Time (OA-OT). The collection of all combinations with the corresponding accuracies (accuracy for each agent) is added to a matrix named as a combinatory matrix (CM) that is shown in Equation (4.3). Figure 4.2 provides a conceptual diagram of our proposed framework for fair reward distribution based on CM.

$$CM = \begin{pmatrix} a_{1,1} & a_{2,1} & \cdot & a_{m,1} & ac_1 \\ a_{1,2} & a_{2,2} & \cdot & a_{2,2} & ac_2 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{1,k} & a_{2,k} & \cdot & a_{m,k} & ac_k \end{pmatrix} \quad (4.3)$$

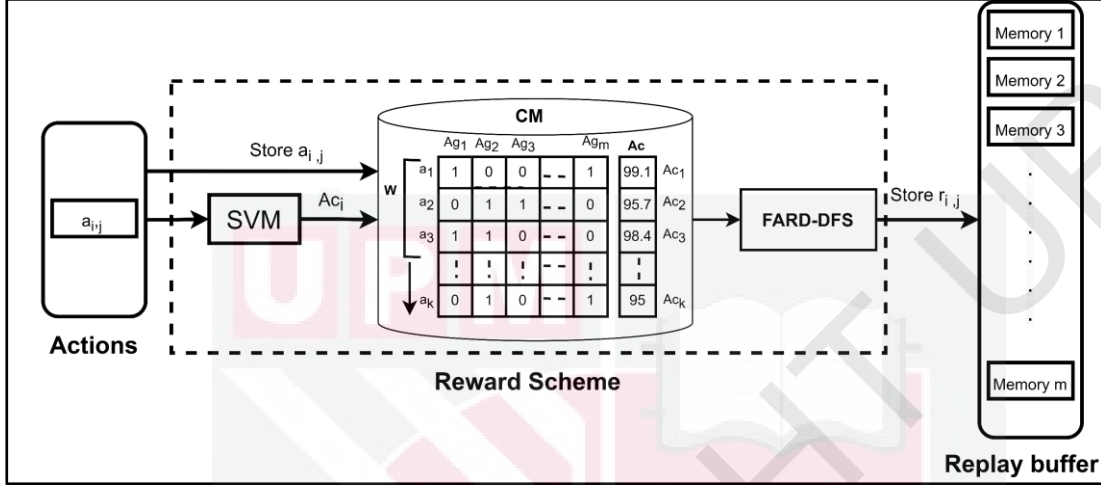


Figure 4.2: Our proposed sub-model for DQN-MAFS, which is used for fair reward distribution to agents and is designated as fair agent reward distribution based dynamic feature selection FARD-DFS

Accumulative Contribution (ACC)

This method is proposed to calculate the reward of each agent based on their importance contribution to the actions taken. In this method, the reward for each agent is calculated as an aggregation of the various decisions multiplied by the aggregation of the accuracies. This is applied when the decision is changing. However, when the decision is fixed and the average accuracy is below certain threshold (Thr), then the reward is discounted with the factor w that denotes the window length, as shown in Equation (4.4). This is achieved by utilizing a sliding window with a specific length size w , the sliding window crosses over the CM to calculate the reward contribution for each agent. It is important to note that the threshold (Thr) is a tuning parameter determined through trial and error. In our experiment, this threshold represents

classification accuracy, set at 60%. This means that selected features must achieve at least 60% accuracy; otherwise, they are considered irrelevant or redundant.

$$r_j = \begin{cases} \frac{\sum_{i=1}^k a_{i,j} \times \sum_{i=1}^k ac_j}{w} & \text{if } \sum_{i=1}^k a_{i,j} == 0 \text{ or } \sum_{i=1}^k a_{i,j} == w \text{ and } \frac{\sum_{i=1}^k ac_j}{w} < Thr \\ \sum_{i=1}^k a_{i,j} \times \sum_{i=1}^k ac_j & \text{otherwise} \end{cases} \quad (4.4)$$

Alternative Contribution (ALC)

The second method proposed in this study for reward distribution is Alternative Contribution (ALC). This method aims to distribute the reward values according to the frequency of decision changes and the relation of this frequency with the accuracy changes. The reward calculation according to this method is done using Equations (4.5) and (4.6). The goal of Equation (4.5) is to measure the effect of accuracy change concerning decision change, and Equation (4.6) is to perform probabilistic scaling of the rewards.

$$r_j = \frac{\sum_{i=1}^K (a_{i,j} - a_{i,j-1})(ac_j - ac_{j-1})}{\sum_{i=1}^K (a_{i,j} - a_{i,j-1})} = \frac{\sum_{i=1}^K da_{i,j} dac_j}{\sum_{i=1}^K da_{i,j}} \quad (4.5)$$

$$sm_j = sm(r_j) = \frac{e^{r_j}}{\sum_{k=1}^M e^{r_k}} \quad (4.6)$$

Where:

$da_{i,j}$ denotes the differentiation of action over time.

dac_j denotes the differentiation of accuracy over time.

sm_j represents softmax normalization.

For more elaboration, we present the calculation of the reward of the agent in the two methods, ACC and ALC, in Figure 4.3. As it is shown, in the case of ACC, the reward is calculated as the multiplication of two areas: the area under the accuracy curve (Ac) and the area under the action curve. The only exception is that when the decision is fixed, the area under the action curve is 0 or 1. In this case, we divide the calculated reward by w , which represents the length of the window. On the other side, in the case of alternative contribution ALC, the reward is represented geometrically by summation of samples from the curve of accuracy derivative dac_j multiplied by the derivative of actions da_i and divided over the derivative of actions. Next, the rewards are scaled by applying the SoftMax function sm , as shown in Equation (4.6).

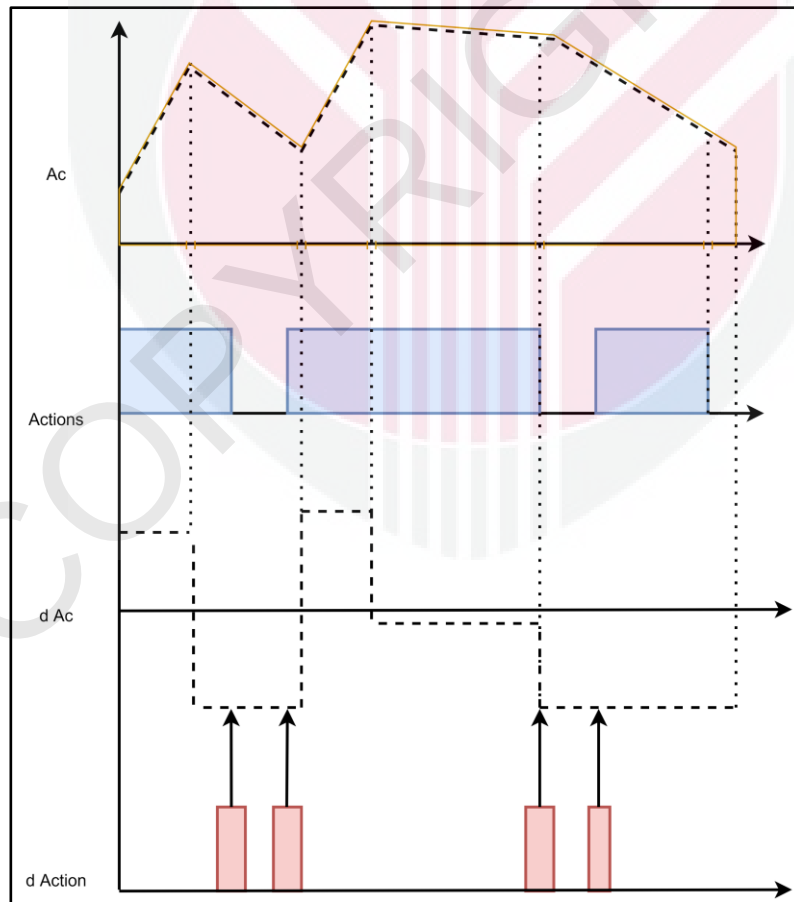


Figure 4.3: Reward calculation for ACC and ALC approaches

Impurity (IM)

The third method of the proposed FARD-DFS uses a regression model based on the selected features and their corresponding labels. This information is considered after training as the importance of features and is used as a reward given the information of other features as well as the information of labels. This method uses random forest (RF) with feature impurity to calculate the reward. We chose the RF because it has a robust built-in feature-importance measurement. In this case, variance reduction (V) is used as a measure of impurity, as shown in Equation (4.7) (Y. Zhang et al., 2012). The lowest impurity is equivalent to a higher reward. Also, this method is based on CM to calculate the impurity of the selected features by agents' action.

$$V = \frac{1}{k} \sum_{i=1}^k (y_i - \mu)^2 \quad (4.7)$$

Where:

y_i is label for an instance, k denotes the number of instances and μ is a value obtained by the mean $\frac{1}{k} \sum_{i=1}^k y_i$.

One Action at One Time (OA-OT)

The last proposed method in this work for reward distribution is named as One-Action at One-Time (OA-OT). This method avoids the challenge of reward distribution by allowing one agent to change a decision at one time. In other words, at one time unit t , it is only permitted for one agent to make a decision change. Mathematically, this is described in Equation (4.8):

$$\forall t, \sum_{i=1}^N |a_i(t) - a_i(t-1)| = 1 \quad (4.8)$$

Where:

N denotes the number of agents, $a_i(t)$ denotes the decision of agent i at moment t .

Hence, the change amount of the accuracy after the action is used as a reward value for the subject agent. To accomplish this, we add a restriction to the combinatory matrix (CM) Equation (4.3) to only have hamming distance between two consecutive rows equal to 1; according to that, a new CM will be generated based on this method and the action policy.

4.2.2.6 Batch Learning for Training DQN

The buffer inside the agent contains a wide range of records that are combined of state, action, next state, and reward. They cannot be used all for training. Hence, a subset of them is used for this purpose. In order to enable effective training, we build a statistical distribution for sampling from the buffer to train DQN. The statistical distribution will give a higher probability for selecting samples subject to misclassification in previous trials. Assuming that the buffer size is N , then the initial probability selection of each sample is $\frac{1}{N}$.

When the sample is selected, it is fed again for evaluation, and a counter $C(index)$ is associated with each sample $index$:

$$C(index) = \begin{cases} C(index) + 1, & \text{if an the reward has increased} \\ C(index), & \text{else} \end{cases} \quad (4.9)$$

Next, the probability of selection is calculated as

$$P(index) = \frac{C(index)}{\sum_{i=1}^M C(i)} \quad (4.10)$$

The increase of the reward is measured by the difference between the current reward $r_{i,t}$ and the reward after training $r_{i,t+1}$ based on the batch and making a decision.

4.2.3 General DQN-MAFS Algorithm

The agents take their own action decisions based on their policy networks $\pi_i : s \rightarrow p(a_i)$. The policy π_i denotes the transition probability that identifies which action a_i should be taken by an agent to evolve the feature sub-space from state s_t to s_{t+1} , and each agent i should receive a reward $r_{i,t}$. For exploring more possibilities for selecting features, we utilize the $\varepsilon - greedy$ policy in this study, as shown in Equation (4.11) (Barto, 2018). Algorithm 1 shows the pseudocode of the $\varepsilon - greedy$ policy.

$$\pi(s_{i,t}, a_{i,t}) = \begin{cases} \text{random } a_{i,t}, & \varepsilon \\ \arg \max_a Q^\pi(s_{i,t}, a_{i,t}; \theta), & 1 - \varepsilon \end{cases} \quad (4.11)$$

Random $a_{i,t}$ denotes the agent has transition probability ε to make random action at state $s_{i,t}$, whilst in case the agent has the transition probability $1 - \varepsilon$ to take action $a_{i,t}$ by exploiting the maximum action value according to the output of the evaluation network that provides the maximum expected reward. This step enables the proposed approach to balance the exploration and exploitation processes (Z. Fang et al., 2019).

Algorithm 1: Epsilon greedy action

Input: (1) expectedReward: list of expected rewards for each possible action
 (2) epsilon: float number between 0 and 1, refers to exploration probability.
Output: (1) action: an integer represents the index of the best action.
Start Algorithm
 1: action = None
 2: **if** random() $\leq$ epsilon **then**
 3: action = randInteger()
 4: **else**
 5: action = argmax(expectedReward)
 6: **end if**
End Algorithm

Thus, in the training DQN process, each agent trains its policy networks via sampling its instances from their replay buffer separately using our proposed model given in Sub-section 4.1.2.6. Then feed them to the DQN for training in order to achieve the maximum action value and maximum long-term expected reward value according to Bellman Equation (4.12) (Barto, 2018).

$$Q(s_{i,t}, a_{i,t}; \theta_t) = r_{i,t} + \gamma \max Q(s_{i,t+1}, a_{i,t+1}; \theta_{t+1}) \quad (4.12)$$

Where θ_t is the parameters of the Q network setting (evaluation network), $\max Q(s_{i,t+1}, a_{i,t+1})$ denotes the maximum action value according to the $\varepsilon - greedy$ policy, and γ represents the discount factor. The sampled transition from the replay buffer $(s_{i,t}, a_{i,t}, s_{i,t+1}, r_{i,t})$ is input into the Q network, in which $s_{i,t}$ is fed into the evaluation network to obtain the Q value $Q(s_{i,t}, a_{i,t}; \theta)$ of the action $a_{i,t}$, while the $s_{i,t+1}$ is inputted to the target network to achieve the maximum Q value $\max Q(s_{i,t+1}, a_{i,t+1}; \theta_{t+1})$. Then the expectation $y_{i,t}$ of the reward is obtained based on the output of the target network via calculating the maximum Q value as demonstrated in Equation (4.13) (Barto, 2018).

$$y_{i,t} = \begin{cases} r_{i,t}, & \text{if step ends at } t + 1 \\ r_{i,t} + \gamma \max Q(s_{i,t+1}, a_{i,t+1}; \theta_{t+1}), & \text{otherwise} \end{cases} \quad (4.13)$$

In order to set the parameters of the Q network, the gradient descent method is used. Thus, the parameters of the Q network can be updated frequently to calculate the Q value accurately. The loss function $L(\theta)$ is calculated according to expectation value $y_{i,t}$, and Q value $Q(s_{i,t}, a_{i,t}; \theta)$ shown in Equation (4.14) (Barto, 2018).

$$L(\theta) = (y_{i,t} - Q(s_{i,t}, a_{i,t}; \theta))^2 \quad (4.14)$$

The evaluation network's parameters are modified by calculating the derivative of the θ as shown in Equation (4.15) (Barto, 2018).

$$\theta \leftarrow \theta + \alpha (y_{i,t} - Q(s_{i,t}, a_{i,t}; \theta)) \nabla Q(s_{i,t}, a_{i,t}; \theta) \quad (4.15)$$

To preserve the stability of the proposed method that may be impacted by the rapid updating of Q network's parameters, the target network's parameters θ_{t+1} are derivative from the parameters of the evaluation network θ at every end of the epoch in training. Hence, after the given number of epochs, the process of parameter updating will be finished.

Algorithm 2: DQN-MAFS algorithm

Input: Chunks of Dataset, list of feature spaces $f = \{f_1, f_2, f_3, \dots, f_m\}$, window w
Output: selectedFeatures
Start Algorithm:
1: Initialize replay buffer size (RB)
2: Initialize multi-agents RL equal to number of features
3: Initialize Q action_value with weights parameter θ for evaluation network
4: Initialize $\hat{Q}$ action_value with weights parameter $\theta_{t+1} \leftarrow \theta$ for target network
5: **for** everyChunk
6: **for** $t = 1$ to T **do**
7: **for** each agent
8: Using a classifier to produce CM based on Eq. (4:3) for selected features based on window w
9: **for** $n = 1$ to N sample in CM **do**
10: Using FARD-DFS approaches to observe the reward r_t for each agent i
11: Assign generated $r_{i,t}$ for each agent based on their actions $a_{i,t}$ and store in RB
12: **end for**
13: Sample batches of transitions($s_{i,t}, a_{i,t}, s_{i,t+1}, r_t$)
14: Calculate the target value of the target network using Eq. (4:13)
15: Perform gradient descent using Eq. (4:14)
16: action $a_t = \operatorname{argmax}_a \hat{Q}(s_{i,t}, a_{i,t}; \theta)$ based on Eq. (4:11)
17: **end for**
18: **end for**
19: Add action to selectedFeatures
20: Update ϵ , reset $\theta_{t+1} \leftarrow \theta$
23: **end for**
End Algorithm

4.2.4 Detector

The last component of our proposed framework is the classification process implemented in this study since it is crucial for predicting XSS attacks. Thus, a set of classifiers are adopted to classify the records based on the selected optimal features provided sequentially from the previous stage DQN-MAFS for each chunk. Then the current knowledge of the classifier is updated with new knowledge for the next chunk at the moment $t + 1$ or whenever a new data stream is available. Based on the literature, we identified four commonly used classifiers for XSS attack detection: Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Decision Tree (DT), and Logistic Regression (LR), as detailed in Chapter 2 (Section 2.7.1). These classifiers were employed for benchmarking to evaluate the performance of the DQN-MAFS model in combination with various FARD-DFS methods. This evaluation allowed us to determine the most effective classifier, which was then selected as the base classifier for our framework.

4.3 Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS)

The second proposed model of evolved XSS attack is a Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS). This stage is proposed by combining both spatial and temporal feature selection in the XSS detection to enable the capability of selecting optimal features in both spatial and temporal dimensions. This model consists of several components, namely, data decomposition, pre-processing, adaptable context analysis, dynamic feature selection and classification, as shown in Figure 4.4. We present each of the components in the subsequent sub-sections.

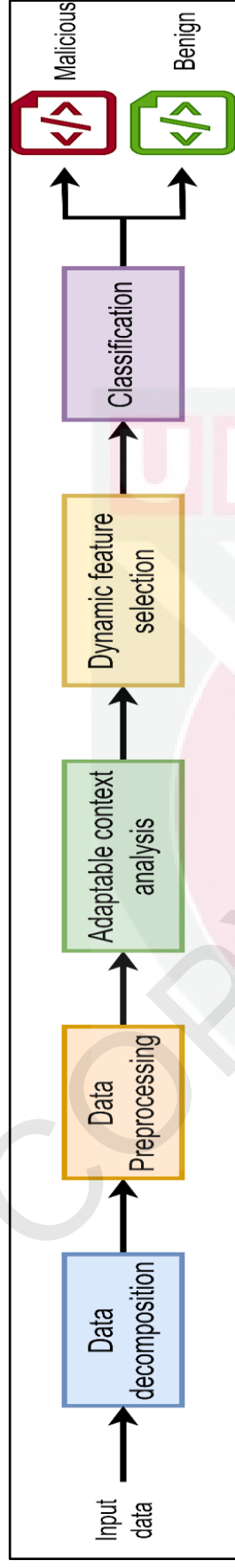


Figure 4.4: Overall 2TS-DFSmodel

4.3.1 Data Decomposition

The data is decomposed into chunks, where each chunk represents a set of records. In other words, considering that no stream dataset available for XSS, we decompose the existing data (JS or payloads) according to their order. Each chunk is constructed by grouping data consecutively without relying on time intervals, as the dataset lacks explicit timestamps. This ordering is essential for the model to grasp the sequential nature of the data effectively. Formally, we write it as in Equation (4.16):

$$D = \{(X_t^c, Y_t^c), t = 1, 2, \dots\} \quad (4.16)$$

Where:

X_t^c : This denotes the samples or feature data that belong to cluster c at time t . Essentially, it is the input features collected in the data chunk t for the specific cluster c .

Y_t^c : This denotes the labels corresponding to the samples X_t^c at time t and cluster c . It represents the target or output labels for the data chunk t within cluster c .

For textual data, the data is reproduced as balanced data by using a resampling method. The mission of resampling is to do under-sampling from the majority class to balance it with the minority class. This is in order to isolate the problem of imbalance from the problem of feature evolving. The former is out of the scope of this research. However, the resampling was not performed for the numerical data which was kept as imbalance because of its simplicity compared with the textual data.

It is important to note that we have indeed considered several factors to ensure the effectiveness of our approach. The chunking process involves dividing the dataset into chunks of roughly uniform size to align with the buffer capacity of the algorithm. This uniformity is crucial for maintaining consistency in processing and analysis. Additionally, we take into account the distribution of data within the dataset. Our approach ensures that each chunk is representative of the overall dataset distribution, thus preserving the integrity of the data and its patterns. By doing so, we account for any potential variations in data distribution that might affect the performance of the feature selection process. The chunking procedure also maintains the ordinal aspect of the data, which is important for retaining the temporal or sequential relationships inherent in the dataset. This consideration helps in accurately capturing the dynamics of feature drift over time.

4.3.2 Preprocessing

The data are one of two types, JS files or payloads. In the case of JS, we use decoding and AST generation. In the case of payloads, we use three steps of pre-processing, namely, decoding, generalization, and segmentation. Each of the steps are presented in the subsequent sections:

4.3.2.1 Decoding

The goal of decoding is to identify the payload or JS more accurately, considering that attackers might exploit the decoding and parsing performed by the browser and utilize encoding techniques to bypass web application firewalls. Hence, decoding is a

simulation of the browser operation in that it decodes and parses the JS and payload to restore the original representation. Furthermore, the encoding process is considered as most the significant strategy among other evading strategies that are used by attackers to mutate their attacks as discussed in Section 2.3.3. At this stage, four common types of decoding are utilized: URL decoding, Base64 decoding, Unicode decoding, and HTML decoding, as detailed by (Y. Fang, Xu, et al., 2020). These decoding methods are summarized in Table 4.1.

Table 4.1: Decoding example for payload and JS malicious files

Decoding type	Encoded samples	After decoding
URL	%253Cimg%2520src%253D1%2520one rror %253Dprompt%25281%2529%253E	
Base64	<iframe src="dat
a: text/html;base64,PHNjcmlwdD5hbGVyd CgxKTWvc2NyaXB0Pg==">	<iframe src="dat
a:text/ html;base64,<script>alert(1)</script >">
Unicode	\u003cimg%20src=1%20onload=alert(12 3)\u003e	<img%20src=1%20onload=alert (123)>
HTML		

4.3.2.2 Generalization

Generalization indicates converting the data to a higher level of representation. The goal of using this in our pre-preprocessing step is to reduce computational complexity, remove noisy information and improve efficiency. Our generalization includes replacing the number that appeared in the data [0, 1, 2, ..., 9] with “0” and replacing the host address in the domain with “http://a”.

4.3.2.3 Segmentation

Segmentation involves the process of breaking down the payload into a collection of individual words. In the context of XSS detection, regular expression tokenizer available in NLTK toolkit (Nature-Language Toolkit) within the Python library, is utilized to partition the payload data into multiple segments. In our method, each segmented keyword is regarded as an individual word (we will refer to these segmented keywords as "word" in the subsequent sections) (Y. Fang, Huang, et al., 2020). The expressions utilized for this segmentation process are detailed in Table 4.2.

Table 4.2: Regularized word segmentation rules for payload

Regular Expression	Splitting Object
(?x)[\w\.\.]+?(	Function
"\w+?"	Contents contained in double quotes
'\w+?'	Contents contained in single quotes
<u>http://\w</u>	URL Links
</\w+>	Termination Tags
<\w+>	Start Tags
\w+=	Window Activations

4.3.2.4 Abstract Syntax Tree

In this stage, the Abstract Syntax Tree (AST) representation is adopted exclusively in the case of JS files. The AST serves as tree-like depiction of the source code structure. This representation plays a crucial role in the semantic analysis, making it a fundamental component in this process. Thus, is adopted within the proposed method for static JS files analysis. To transform JS source code into an AST, the Esprima parser (Hidayat, 2018) is utilized. Esprima stands as a high-performance JS parser capable of taking a JS code in the form of a string and generating a corresponding syntax tree representation. For illustration, following JS code is considered an

example, Esprima parses this code and converts it into an abstract syntax tree. Subsequently, this abstract syntax tree undergoes a depth-first traversal to generate sequences of syntactic units. An example of producing AST from a small JS file is presented in Figure 4.5.

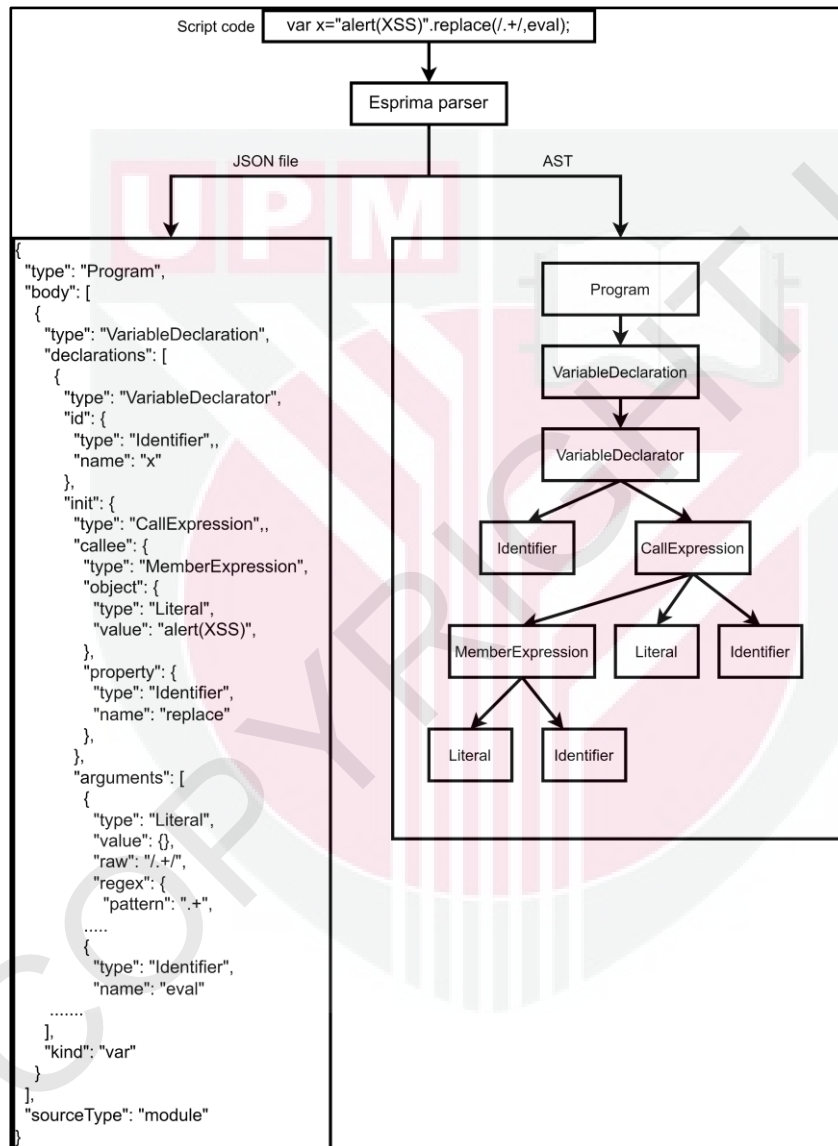


Figure 4.5: An Example of generating an Abstract Syntax Tree (AST)

4.3.3 Adaptable Context Analysis

The role of adaptable context analysis is to extract a suitable dynamic vector representation that efficiently downstream the evolved XSS attacks detection. Such dynamic word vectors carry real context information, unlike traditional static word embeddings (Peters et al., 2017). In this model, we extract the syntactic unit sequences with detailed information from the AST nodes and segmented words for JS files and payload records respectively. These sequences are then transformed into dynamic vector representations. To achieve this feature representation, the dynamic word embeddings model is adopted which encompasses Word2Vec (Mikolov et al., 2013a) and a two-layers of Bi-LSTM network. This model can derive dynamic word vectors through contextual analysis of syntactic unit sequences. Notably, the same syntax unit appearing within various contexts is represented by distinct word vectors, effectively resolving the polysemy in code. Figure 4.6 illustrates the utilized adaptable context analysis model, which is similar to approaches employed in these previous studies (Y. Huang et al., 2021; Peters et al., 2017). The subsequent sections delve into details of each of the sub-components of this model.

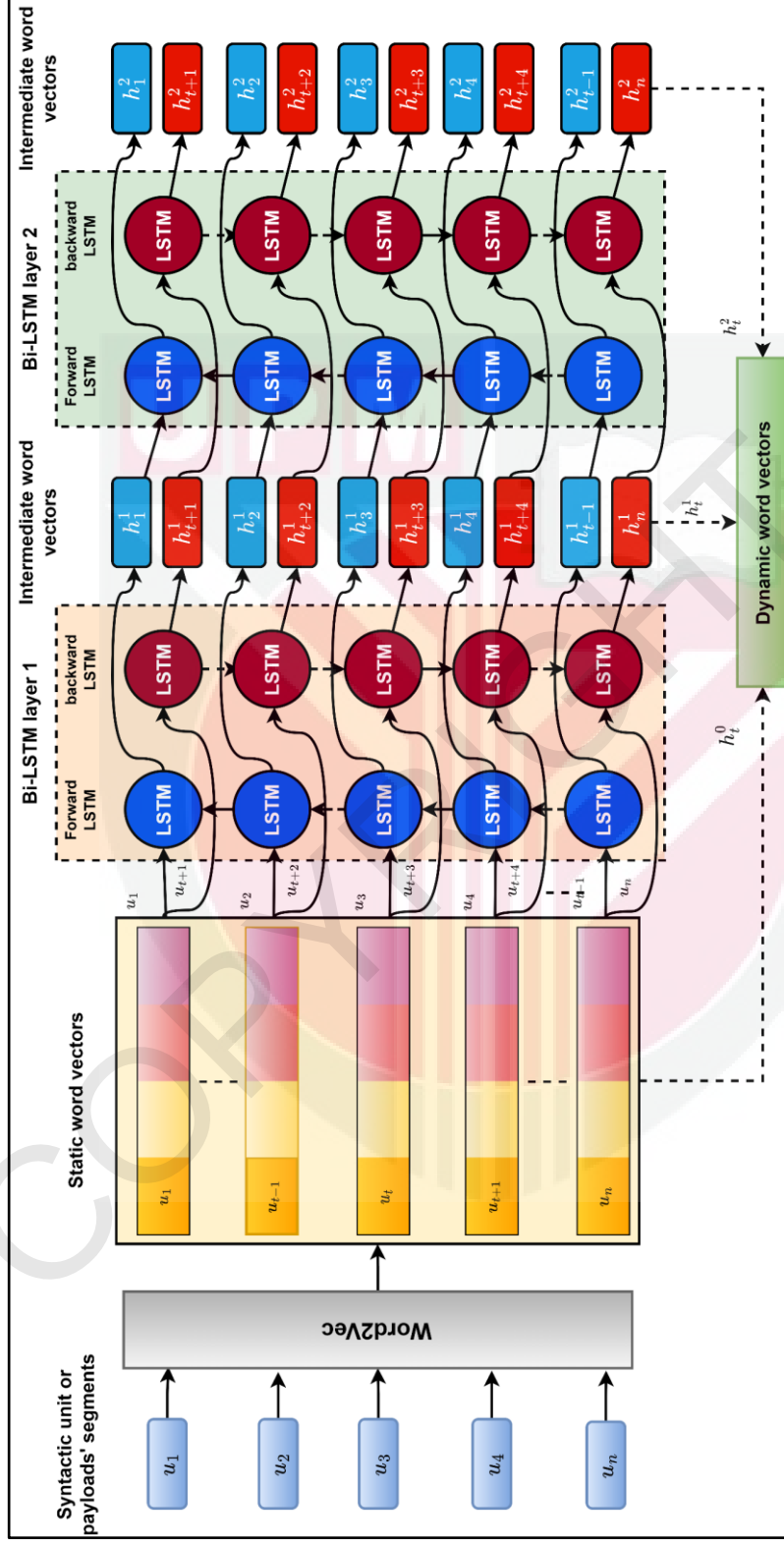


Figure 4.6: Dynamic word embedding approach overview

4.3.3.1 Syntactic Unit Sequences Extraction

After preprocessing the payload and JavaScript (JS) files, the payload is segmented, and an Abstract Syntax Tree (AST) is generated by parsing the JS files. At this stage, the syntactic units can then be extracted.

As depicted in Figure 4.5, JS file undergoes parsing using Esprima, resulting in a representation in JavaScript object notation (JSON) file format that has several lines. Each line of the JSON file corresponds to a syntax unit, as explained in Section 4.3.2.4 the special symbols are removed from each syntax unit to create input for Word2vec and Bi-LSTM.

Utilizing Esprima, a total of 69 distinct node types are generated, including the root node of type Program, Statement type nodes, Variable_Declaration nodes that indicate the variable declarations, Variable_Declarator nodes providing descriptions of the variable declarations, Identifier nodes used for variable naming, and Call_Expression nodes that represent function-call expressions. Furthermore, there are nodes like Member_Expression, which represent references to object members, and Literal nodes that denote literal values, encompassing strings, Booleans, numbers, null values, regular expressions, and more (Y. Fang, Huang, et al., 2020). By performing a depth-first traversal of the nodes within AST, we can ultimately extract the text sequences of syntactic units for subsequent training process, as demonstrated in Table 4.3. It is worth noting that different code snippets are mapped into distinct types of nodes, which we also refer to as syntactic unit. It is important to highlight that this study treats each node within AST a syntactic unit, in contrast to the approach employed by (Y. Huang et al., 2021), who considered each line within the AST as a unit.

Table 4.3: An example of extracted syntactic units from the AST

No.	Syntactic node type
1	Program
2	Variable_Declaration
3	Variable_Declarator
4	Call_Expression
5	Identifier
6	Literal
7	Member_Expression
8	Identifier
9	Literal

Similar to the earlier method, the Syntactic units are also extracted from the payload in case of the XSS attack in the form of a payload that can be injected with URL link, request HTTP, or response HTTP messages, without generating the AST. After performing the preprocessing steps including decoding, generalization, and segmentation, the Syntactic units can be extracted directly from the output of these steps in which each single segmented word represents a single Syntactic unit.

4.3.3.2 Extraction of Word Embedding using Word2Vec

Upon obtaining the sequences of syntactic units, the next step involves representing these sequences for feature learning and classification techniques. To make them compatible with learning models, these sequences must be transformed into a format that can be processed, specifically, word vectors. It is crucial to note that the effectiveness of the deep learning models largely depends on the outcome of the word vector model training. Thus, determining the most suitable word vector model for the corpus utilized in this study becomes imperative.

Word2Vec stands out as one of the most prevailing methods employed in natural language processing (NLP) for acquiring word embeddings (Mikolov et al., 2013b).

Word embeddings represent words as dense vectors in a high-dimensional space, in which words with similar meanings are either identical or positioned close to one another. This transformation converts the processed textual content into vectors within a K-dimensional vector space, leveraging the similarity within this vector space to denote the semantic resemblances among texts. The Word2Vec model primarily encompasses two training models, namely, the Skip-gram model and the Continuous-Bag-of-Words (CBOW) model, illustrated in Figure 4.7. Both these models transformed into word vectors that consider the proximity of words to the central word. In essence, these models retain certain word order relationships within the context window but do not explicitly capture the precise order or distance between individual words.

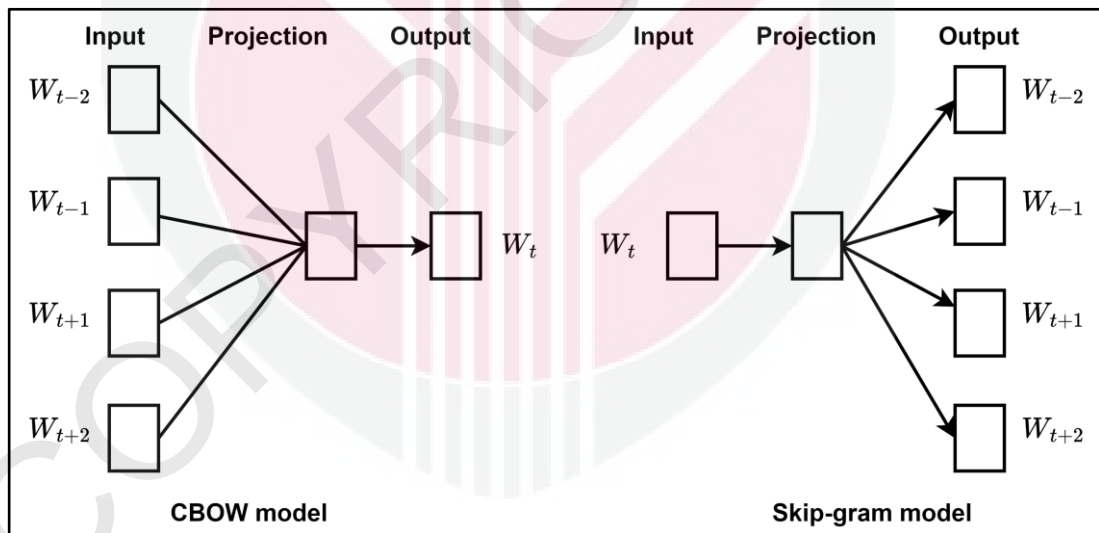


Figure 4.7: Skip-gram and CBOW models

In the CBOW model, the sliding window shifts once during each training step to generate a training sample, meaning it predicts the targeted word relied on its context. This model adjusts word vectors uniformly with the words surrounding them, such way is not as effective in training infrequent words. On the other hand, the skip-gram

model also shifts the sliding window once during each training step, but it generates multiple training samples, predicting the context of the targeted word from its surroundings. The skip-gram model requires multiple adjustments to train its word vectors and performs better with smaller corpus and rare words. Although the skip-gram model costs longer to train compared to CBOW, but it yields relatively superior results. Consequently, the experiment opts to employ the Skip-gram model for training the XSS word vector.

The Skip-gram model comprises three components: the input layer, the hidden layer, and the output layer. The central word, fed into the input layer, undergoes transformation through the projection of the hidden layer to reveal its context window in the output layer. The weights of the hidden layer are acquired through learning from the training data in the Skip-gram model, resulting in the final representation of word vectors. The main aim of the Skip-gram model is to discover word representations that prove valuable in predicting the words that appear around a given word within a sentence or a document. In more formally, when provided with sequences of training words (syntactic units) $w_1, w_2, w_3, \dots, w_T$, the Skip-gram model aims to maximize the average log probability, as outlined in Equation (4.17) (Mikolov et al., 2013b).

$$\frac{1}{T} * \sum_{t=1}^T * \sum_{(-c \leq j \leq c, j \neq 0)} \log p(w_{(t+j)} | w_t) \quad (4.17)$$

Where ' c ' denotes the size of the training context, which can be function dependent on the central word w_t . A larger ' c ' leads to more training examples, potentially resulting in enhancement of the prediction accuracy, albeit with increased training time. The

formulation of Skip-gram describes $p(w_{t+j} | w_t)$ using a SoftMax function, as depicted in Equation (4.18) (Mikolov et al., 2013b).

$$p(w_o | w_I) = \frac{\exp(\hat{v}_{w_o}^T \cdot v_{w_I})}{\sum_{w=1}^W \exp(\hat{v}_{w_o}^T \cdot v_{w_I})} \quad (4.18)$$

Where v_w and $\hat{v}_w$ represent the input and output representations of given word w , respectively, while W denotes the total number of words within the vocabulary dictionary.

To illustrate the process of this stage, an example of a malicious XSS sample is taken, “topic=http://u<script>alert(document.cookie)</script>”. After the segmentation process, this sample will be as (“topic=”, “http://u”, “<script>”, “alert”, “(”, “document.cookie”, “)”, “</script>”). Consider the assumption that the “alert” is inserted in syntactic unit as our input segmented word. Also, the sliding window size is set as 3, meaning that three words from either the left or right side of the selected input word. Then the words within the context window (with the selected input word) are {“alert”, “(”, “document.cookie”, “)”} from the right side. If the sliding window moves to left side, the words within the context window (also with the selected input word) become {“topic=”, “http://u”, “<script>”, “alert”}. In our experiments, we have set the window size to 8, as illustrated in Section 4.3.3.2.

4.3.3.3 Dynamic Word Embedding Extraction

In NLP, the meaning of a word often depends on the words that come before and after it. Thus, some researchers choose to use a bidirectional LSTM (Bi-LSTM) instead of

a regular LSTM algorithm for semantic feature extraction because Bi-LSTM can capture both forward and backward contexts in a sequence, whereas regular LSTM can only capture forward context. Another advantage of Bi-LSTM is that it can capture long-term dependencies in a sequence, which can be difficult for regular LSTMs. This is because Bi-LSTM has two separate hidden states, one for the forward pass and one for the backward pass, which can help it better model complex dependencies in the sequence.

Word2Vec can generate the static word embedding representation by representing words based on their positional relationships in K-dimensional space, thereby capturing the semantic relationship among them. However, to incorporate more contextual information, we can use two Bidirectional-LSTMs (Bi-LSTMs) as an additional feature extraction step to obtain dynamic word embedding representations that is represented mathematically in Equations (4.19-4.23) and similar to (Y. Huang et al., 2021).

More formally, given a sequence of N syntactic units, $(u_1, u_2, u_3, \dots, u_N)$, a forward language model (two-layers LSTM) computes the probability of syntactic unit u_s by given the previous syntactic units $(u_1, u_2, u_3, \dots, u_{s-1})$ as represented Equation (4.19), the probability of the sequence is also calculated using this method. After the classical static word embedding model Word2Vec is used to generate its representation w_s then pass it through two layers of forward LSTMs. At each syntactic unit s , each LSTM layer output an intermediate word vector representation $\tilde{h}_s^j (j = 1, 2)$ with context information.

$$p(u_1, u_2, u_3, \dots, u_N) = \prod_{s=1}^N (u_s | u_1, u_2, u_3, \dots, u_{s-1}) \quad (4.19)$$

Similar to the forward language model, the backward language model computes the probability of the current syntactic unit by giving following syntactic units. Similarly, at each syntactic unit s , each LSTM layer output an intermediate word vector representation $\tilde{h}_s^j$ of u_s by given the future context $(u_s | u_{s+1}, u_{s+2}, u_{s+3}, \dots, u_N)$ as Equation (4.20).

$$p(u_1, u_2, u_3, \dots, u_N) = \prod_{s=1}^N (u_s | u_{s+1}, u_{s+2}, u_{s+3}, \dots, u_N) \quad (4.20)$$

Dynamic word embedding model combines forward and backward language models. As Equation (4.21), jointly maximizes the log likelihood function of bidirectional model. Where Θ_x is the parameter for syntactic unit representation layer and Θ_s is the parameter for Softmax layer, which shared in the forward and backward language models, except for the LSTM parameters (θ_{LSTM}). In fact, it is equivalent to training a forward and backward language model, respectively.

$$\sum_{k=1}^N (\log p(u_s | u_1, u_2, u_3, \dots, u_{s-1}; \Theta_x, \tilde{\theta}_{LSTM}, \Theta_s) + \log p(u_s | u_{s+1}, u_{s+2}, u_{s+3}, \dots, u_N; \Theta_x, \vec{\theta}_{LSTM}, \Theta_s)) \quad (4.21)$$

For each syntactic unit u_s , a bidirectional language model computes a set of $2L + 1$ representation, where L represents the layer number of the bidirectional language model, h_s^0 is the static word embedding layer and $h_s^j = [\tilde{h}_s^j; \vec{h}_s^j]$ as Equation (4.22).

$$R_s = \{x_s, \tilde{h}_s^j, \vec{h}_s^j | j = 1, 2\} = \{h_s^j | j = 0, 1, 2\} \quad (4.22)$$

4.3.3.4 Weighted Sum Combination

Different layers of the model learn different information, so it is necessary to calculate the weighted combination of the three features. Considering the downstream classification task, we fold all the layers in R into a vector and use V to represent this dynamic word vector. We compute the task-specific weights for all the bidirectional language model layers as Equation (4.23), s_j are softmax-normalized weights that indicates how much attention should be placed on each layer and the scalar parameter γ is a global scaling factor that allows the task model to scale the entire V vector.

$$V_s = E(R_s; \Theta) = \gamma \sum_{j=0}^L s_j h_s^j \quad (4.23)$$

4.3.4 Dynamic Feature Selection

In order to enable dynamic feature selection that accomplish both spatial and temporal feature drift handling, we conduct local searching algorithm that starts from an initial solution, perform certain operation to alter the solution, accepting the new solution in case of improvement or iterating again the operation to alter the solution, and repeating the process until reaching an improved solution. The process is repeated for a pre-defined number of iterations and the best obtained solution is generated as the best feature selection solution. Referring to the matrix M_t where $M_t \in \{0,1\}^{m_t \times n}$ that is defined in the problem formulation in Chapter 3 (Section 3.3), this matrix represents the solution structure. The fitness function is defined as the validation accuracy and F1-measure of the decision of feature selection.

Data is inputted to the proposed model as chunks with a different time stamp for each chunk, then it is preprocessed to extract hidden state, after that presented to 2TS-DFS. After receiving a chunk, it will generate a mask with the same size of the chunk that contains only zeros and ones. The value one indicates selecting the corresponding feature and the value zero indicates deselecting the corresponding feature. As shown in Figure 4.8, the internal process of 2TS-DFS is combined of several steps: 1- clustering of the records inside the chunk and grouping them again into clusters, as detailed in Subsection --. 2- Generate the solution representation according to the grouping resulted from step 2. 3- Provide the solution representation to TS for conducting the searching and finding the best solution. 4- enabling the best solution as a feature selection decision and generating a reduced number of features.

4.3.4.1 Clustering Building Using Agglomerative Clustering

Classifying malicious XSS attacks using raw payload or JS data in a data stream can be a challenging task, as the data is constantly changing and evolving over time. This resulted in high differences in the underlying structure of the data, which can make it difficult for a classical ML or DL model to capture all of the variation present in the data. Thus, the use of clustering as a preprocessing step can be beneficial in this context, as it can help identify patterns and group together similar data points that have a more homogenous structure in real-time. Agglomerative clustering is a popular hierarchical clustering algorithm due to its simplicity and ability to handle large datasets (Tokuda et al., 2022). The details of agglomerative clustering have been presented in Chapter 2 (Section 2.2.10).

Let the distance between any two clusters w and v be represented as $D(w, v)$. Many distinct metrics can be used for defining D , but here we use Euclidean distances. Initially, TS treats each single object as singleton cluster, that is, the number of clusters is given by the number of labelled records within the chunk (bottom-up approach). At each iteration, pairs of clusters that were not previously joined are inspected, and the pair of clusters having the minimum value in D is joined to form a new cluster u . Let s and t be the pair of clusters that were joined, and define $x[i]$ to represent a record i in cluster x . After the clusters are joined, the distances $D(u, v)$ between the new cluster u and all other clusters need to be calculated. We used the Ward linkage (Ward, 1963) to calculate the new distances between clusters. Ward linkage joins clusters leading to the minimum increase in within-cluster variance (Tokuda et al., 2022). This can be done by setting the distance as follows in Equation (4.24).

$$D(u, v) = \sqrt{\frac{|v|+|s|}{T} \cdot D(v, s)^2 + \frac{|v|+|t|}{T} \cdot D(v, t)^2 - \frac{|v|}{T} \cdot D(v, t)^2} \quad (4.24)$$

Where $T = |v| + |s| + |t|$.

4.3.4.2 Solution Representation

After the clustering, the solution representation is generated. Assuming that the clustering has resulted in N_c clusters that have the original number of features of the records is N_f . Based on this, the solution is represented by a binary matrix $S = [s_{ij}]$ with the dimension of $N_c \times N_f$ where $s_{ij} \in \{0,1\}$. $s_{ij} = 0$ indicates that the feature j in the cluster i is not relevant while $s_{ij} = 1$ indicates that the feature j in the cluster i

is relevant. Hence, the searching algorithm presented in the next sub-section is responsible on searching with the space of $\{0,1\}^{N_c \times N_f}$ to maximize the objective function which represents the accuracy.

4.3.4.3 Tabu Search Algorithm

Afterwards, we call the algorithm of Tabu search with the configuration of generating neighbors as a purely random solution. This assists at exploring the solution space more efficiently due to the high dimension of the solution space. Each element in the solution is generated based on generating random number r between 0 and 1 from uniform distribution, comparing its value with 0.5. In the case of $r > 0.5$ them we assign 1 and in the case of $r \leq 0.5$ we assign 0. The optimal solution after calling Tabu search is S^* .

The implementation of Tabu search for feature selection is shown in following steps:

1. **Initialization:** Let x_0 be the initial solution, where $x_0 \subseteq X$ that represents the solution space.
2. **Evaluation:** Evaluate the fitness of the current subset solution x_i based on acc and F1 score of a classification model trained on x_i as shown in Equation (4.25):

$$f(x_i) = w_{acc} * acc_{x_i} + w_{f1} * f1_{x_i} \quad (4.25)$$

where acc_{x_i} and $f1_{x_i}$ are the accuracy and F1 score of a classification model trained on x_i , respectively, and w_{acc} and w_{f1} are weights that control the relative importance of accuracy and F1 score.

1. Neighborhood generation:

Generate a set of neighboring subsets of features $N(x_i)$ by making small changes to the current subset of features x_i . The changes are five bits for each row in the solution as shown in Algorithm 3 pseudocode. The bits will be flipped from zero to one or from one to zero.

2. Tabu search:

Select the best neighboring subset of features $x_i \in N(x_i)$ based on the objective function $f(x_i)$ and the constraints imposed by the Tabu list TL . The Tabu list is a set of previously visited subsets of features, and $TL \subseteq 2^x$ is the set of all possible subsets of features that are currently Tabu. The algorithm selects x_i by solving the following optimization problem as shown in Equation (4.26):

$$x_i = \operatorname{argmax} \{f(x_k): x_k \in N(x_i) \setminus TL\} \quad (4.26)$$

The Tabu list TL is updated after each iteration to include x_i and exclude subsets that have been in the Tabu list for a certain number of iterations.

1. Termination:

Repeat steps 2-4 until both the accuracy and F1 score of the selected subset of features exceed a predefined threshold as demonstrated in Equation (4.27):

$$\operatorname{acc}(x_i) > \operatorname{thr}_{acc} \text{ and } f1(x_i) > \operatorname{thr}_{f1} \quad (4.27)$$

where thr_{acc} and thr_{f1} are the accuracy and F1 score thresholds, respectively.

Algorithm 3: Generate neighbor solution.

Input: current_solution

Output: neighbor_solution

Start Algorithm

01: Start with an initial solution

02: Generate a new neighbor solution using the `neighbour\_solution` function, passing the input solution and an optional `e` parameter (Where e is a Boolean variable indicating that the previous solution was good or not).

03: Check if the new neighbor solution is in the tabu list by calling a function called `Tabu\_list\_checker`.

04: If the new neighbor solution is in the tabu list (indicated by `t` being `True`), go back to step 2 and generate another neighbor solution.

05: If the new neighbor solution is not in the tabu list (indicated by `t` being `False`), return the neighbor solution.

End Algorithm

4.3.4.4 Cluster Assignment

The role of the cluster assignment is to assign any testing record to one of the generated clusters based on the similarity maximization method in the training process. In other words, assigning the record to the cluster that has the highest similarity with compared other clusters. An issue with calculating the similarity between the records, is the high dimension of them. Because each record consists of $N_F = 100$ features that is not a scalar value but an entire vector with a dimension 1×2048 that represents part of the hidden state. In order to resolve this, we convert the vectors from 1×2048 dimension to 1×5 that includes five information, namely, minimum, maximum, mean, standard deviation, and median. Afterwards, the similarity is calculated based on dimension of 5×100 that represents a reduced matrix from the hidden state. The similarity is calculated based on cosine similarity as depicted in Equation (4.28).

$$\text{cosineSim}(x_1, x_2) = \frac{x_1 \cdot x_2}{|x_1| |x_2|} \quad (4.28)$$

Assuming that the clusters set is $C = \{C_i\}, i = 1..N_c$. Then the sample x is added to the cluster based on the Equation (4.29).

$$C_* = \operatorname{argmax}(\operatorname{cosineSim}(x, C)) \text{ where } C_* \in C. \quad (4.29)$$

4.3.4.5 Feature Selection

Once the sample is added to its cluster, a feature selection decision is generated to be the subset of the best solution found by Tabu search that is corresponding to the cluster. This is represented by the Equation (4.30).

$$F^* = S^*(C_*, :) \text{ where } F^* \text{ has the dimension of } 1 \times N_F. S^* \quad (4.30)$$

The results of the feature selection F^* is interpreted by preserving the feature that is corresponding to the value one and converting the feature that is corresponding to value zero to zero.

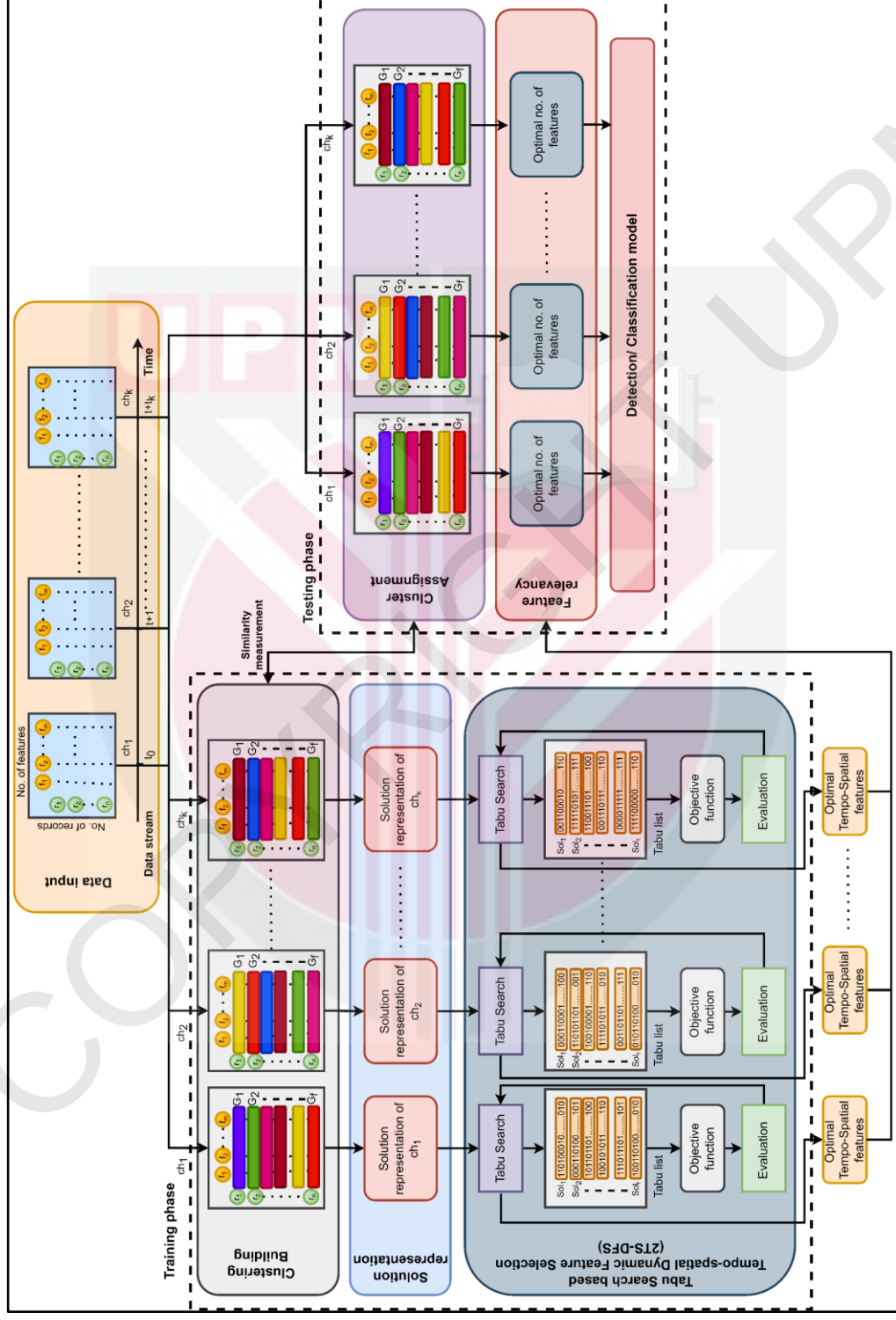


Figure 4.8: Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS)

4.3.5 Classification

Malicious code is a payload vector or only some part of the JavaScript file, so classification model should make full use of this characteristic. TextCNN (Y. Kim, 2014) is a pre-trained CNN model for text classification model suitable that supports variable length text, in which the core idea is to extract key features with varying window lengths. This processes of the TextCNN are represented mathematically in Equations (4.31-4.34) (Y. Kim, 2014).

The framework for TextCNN is based on the CNN algorithm as illustrated in Chapter 2 (Section 2.9.5.1), $X_i \in \mathbb{R}^k$ represents the k-dimensional of dynamic word vector representation corresponding to the s^{th} syntactic unit in the sequence (words). The syntactic unit sequence with length n is represented as Equation (4.31), where $\oplus$ is the concatenation operator.

$$x_{1:n} = x_1 \oplus x_2 \oplus \dots \oplus x_n \quad (4.31)$$

filter $w \in \mathbb{R}^{hk}$ is a key part of convolution operation, used to produce a new feature in a window of h syntactic units (words). This filter is applied to each possible window of the dynamic word vector to produce a feature map. For example, it produces a new feature c_s in a window of $x_{s:s+h-1}$ words by Equation (4.32), where $b \in \mathbb{R}$ is a bias term and f is a non-linear function such as hyperbolic tangent (Gong & Ji, 2018; Y. Huang et al., 2021).

$$c_i = f(w \cdot x_{s:s+h-1} + b) \quad (4.32)$$

This filter slides down the matrix of the sequences $\{X_{1:h}, X_{1:h}, \dots, X_{n-h+1:n}\}$ to produce a feature map $c \in \mathbb{R}^{n-h+1}$ as Equation (4.33). Further, perform max-over-time pooling operation (Collobert et al., 2011) on feature map and take the maximum value $\hat{c} = \max\{c\}$ as the feature representation map corresponding to this filter.

$$c = [c_1, c_2, \dots, c_{n-h+1}] \quad (4.33)$$

This is the process of extracting a feature from a filter. Different filters of TextCNN can obtain different features. Finally, a fully connected layer with Softmax (sm) function use these features to compute the probability distribution over labels by Equation (4.34).

$$sm_j = sm(r_j) = \frac{e^{r_j}}{\sum_{m=1}^K e^{r_m}} \quad (4.34)$$

Where:

r_j denotes input vector.

e^{r_j} denotes standard exponential function for input vector.

K denotes the number of classes.

e^{r_m} denotes standard exponential function for output vector.

4.3.6 2TS-DFS Algorithm

Firstly, the number of chunks, number of clusters, tabu tenure, and the tabu list are received as inputs. An Agglomerative Clustering object is initialized using the specified number of clusters. The algorithm is prepared to produce solutions, training F1 score, training accuracy, test F1 score, and test accuracy as outputs. The algorithm

begins by iterating over each chunk. For each chunk, the training and testing data (X_{train} , y_{train} , X_{test} , y_{test}) for that chunk is loaded. The number of iterations is calculated as 10 times the second dimension of X_{train} . If the chunk number is 0, a new model is created (TextCNN model) and saved. Otherwise, the tabu list, agglomerative clustering, and model from the previous iteration are loaded. The algorithm then extracts statistical information from X_{train} and fits and predicts the Agglomerative Clustering model with this information, producing training clusters. This process is repeated with X_{test} to produce testing clusters. The current solution is initialized as a random binary matrix with dimensions equal to the number of clusters by the second dimension of X_{train} . The solution, F1 scores, and accuracies for both the training and testing data are initialized as empty lists. The algorithm enters a loop that continues for the specified number of iterations. During each iteration, a mask is applied to X_{train} using the current solution and the training clusters. The model is then trained with this masked data. The same mask is applied to X_{test} and predictions are made on the masked data. The F1 score and accuracy of the predictions on the test data are evaluated and saved. The model then makes predictions on the training data, and the F1-score and accuracy are evaluated and saved. The tabu list is updated with the current solution and the tabu tenure. An aspiration criterion is checked, which returns a Boolean value. If the aspiration criteria are not met, the model is loaded again. Otherwise, the current state of the model is saved. The current solution, and the F1 scores and accuracies for both the training and test data, are then added to their respective lists. This process continues until all iterations are completed, at which point the algorithm returns the solutions, F1 scores, and accuracies for both the training and test data. Algorithm 4 presents the pseudocode of the 2TS-DFS algorithm.

Algorithm 4: 2TS-DFS algorithm

```
Input: No. of chunk, No. of cluster, Tabu tenure, Tabu list
        AgglomerativeClustering= AgglomerativeClustering(No. of clusters, affinity='euclidean')
Output: Solutions, F1_train, Ac_train, F1_test, Ac_test
1: Start algorithm
2: for each chunk do
3:   load (X_train, y_train, X_test, y_test) for chunk number i
4:   No. of iteration=X_train.shape[2]*10 (where the shape of X_train is (1600, 2048, 100))
5:   if no. of chunk=0 then
6:     model=create_model() (TextCNN model)
7:     model.save()
8:   else
9:     Tabu_list= load (Tabu_list_i)
10:    agglomerativeClustering= load ()
11:    model= load_model()
12:  End if
13:  X= get statistical_information (X_train)
14:  cluster_train= agglomerativeClustering.fit_predict(X)
15:  cluster_test= get_euclidean_distance(X_test, X_train, cluster_train)
16:  current_solution=randint(low=0, high=2, size= (No. of clusters, X_train.shape[2]))
17:  solutions, F1_train, Ac_train, F1_test, AC_test= Empty_Lists
18:  For step = 0 to No. of iteration do
19:    x_train= apply_mask(X_train, current_solution, cluster_train)
20:    history= model.fit(x_train, y_train)
21:    x_test= apply_mask(x_test, current_solution, cluster_test)
22:    y_pre= model.predict(x_test)
23:    f1_test, ac_test= evaluate_result(y_pre, y_test)
24:    y_pre= model.predict(x_train)
25:    f1_train, ac_train= evaluate_result(y_pre, y_train)
26:    Tabu_list= update_tabu_list(current_solution, Tabu_tenure)
27:    aspiration_criteria = check_aspiration_criteria(f1_train, Ac_train)
    (aspiration_criterial is a Boolean value)
28:    if not aspiration_criteria
29:      load model()
30:    else
31:      model. save()
32:    End if
33:    solutions.add(current_solution)
34:    F1_train.add(f1_train)
35:    F1_test.add(f1_test)
36:    Ac_train.add(ac_train)
37:    Ac_test.add(ac_test)
38:    Save (solutions, F1_train, Ac_train, Ac_test, F1_test)
39:    if F1_train>90 and Ac_train >90 and have best_solution:
40:      Current_solution= generate_neighbour_solution(current_solution)
41:      Best_solution= True
42:    else
43:      current_solution=randint(low=0, high=2, size= (No. of clusters,
X_train.shape[2]))
44:    End if
45:  End for
46: End for
47: End Algorithm
```

4.4 Evaluation Methodology

In this section, the evaluation metrics, datasets, and experimental design are introduced.

4.4.1 Evaluation Metrics

The comparisons are evaluated in terms of classification performance metrics, including accuracy, precision, recall, and F1-measure. Additionally, the number of selected features or feature reduction rate (FRR) are used as memory enhancement measures. All classification metrics are based on confusion matrix parameters as demonstrated in Table 4.4. The confusion matrix parameters are true positive (TP): represents the number of sampled are recognized correctly as an XSS attacks, true negative (TN): signals the number of samples are classified as a benign, false positive (FP): denotes the number of samples are misclassified as a XSS attack, and false negative (FN): refers to the number of XSS attack samples that undetected by the system. Thus, the commonly used metrics are accuracy, recall that is also known as true positive rate (TP) or detection rate, precision, F1-measure, as well as False Positive Rate (FPR).

Table 4.4: Confusion Matrix parameters

		Actual Class	
		Class A	Class B
Predicted Class	Class A	TP	FP
	Class B	FN	TN

Accuracy (Acc): is a proportion of the samples classified correctly to the total number of classified samples. More formally, this value is defined as the number of true-positives plus the number of true-negatives divided by the total of the data points. For example, the many of the classifiers applied here can be measured according to whether they classified malicious XSS as malicious and benign samples as benign. This accuracy rate is computed by Equation (4.35).

$$Acc = \frac{TP+TN}{(TP+TN+FP+FN)} \quad (4.35)$$

Recall (Sensitivity): is also known as True positive rate (TPR), or Detection rate). Recall is a metric that denotes the ratio of the number of positive samples that were correctly predicted (TP) to the overall classifications number in the same actual class. In other words, the percentage of XSS attacks classified correctly to the total of the number of inputs XSS attack samples. Recall metric is used to demonstrate the FN of the model. The Recall can be calculated by Equation (4.36).

$$Recall = \frac{TP}{(TP+FN)} \quad (4.36)$$

Precision: represents the proportion of numbers of the samples that are predicted correctly to the overall number of positive samples. Such a metric is used to show the percentage of the FP rates and can be computed by using the following Equation (4.37). Thus, if there are only true positives, but no false positives, then the precision is 1 (or 100%).

$$Precision = \frac{TP}{(TP+FP)} \quad (4.37)$$

F1-measure: is a measure that combines the recall and precision metrics in a single measure. F1-measure is also called as a harmonic mean of earlier two metrics (Recall and Precision). The F1-measure mostly is used as an evaluation metric for imbalanced datasets and can be calculated by Equation (4.38).

$$F1 - measure = 2 \times \frac{(Recall \times Precision)}{(Recall + Precision)} = \frac{2TP}{2(TP + FP + FN)} \quad (4.38)$$

FPR (False Positive Rate): It is the ratio of all benign samples which are incorrectly detected as malicious as computed in Equation (4.39). FPR was the most often used performance measure in XSS attack detection, Lower FPR indicates better detection performance.

$$FPR = \frac{FP}{TN + FP} \quad (4.39)$$

Feature Reduction Ratio (FRR): Feature reduction ratio is a time series that measures the effectiveness of feature selection. In order to measure the feature reduction ratio, we calculate the complementary ratio of the number of selected features over the number of features in the original data to the value 1 as computed in Equation (4.40).

$$FRR_t = 1 - \frac{N_{s,t}}{N_{o,t}} \quad (4.40)$$

Where:

$N_{o,t}$ denotes the number of features in the original data at moment t

$N_{s,t}$ denotes the number of selected features at moment t

4.4.2 Statistical analysis

To substantiate the effectiveness of the proposed methods compared to benchmark methods, A various statistical significance tests on the accuracy and F1-measure results were conducted. The employed tests are the t-test, Shapiro-Wilk test, and Friedman test.

t-test: The t-test is used to determine if there is a significant difference between the means of two groups. There are different types of t-tests, such as the independent samples t-test and the paired samples t-test. Here, the independent samples t-test is calculated as represented in equation (4.41) (Field, 2009; Potochnik et al., 2018):

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \quad (4.41)$$

where:

$\bar{X}_1$ and $\bar{X}_2$ are the sample means of groups 1 and 2, respectively.

s_1^2 and s_2^2 are the sample variances of groups 1 and 2, respectively.

n_1 and n_2 are the sample sizes of groups 1 and 2, respectively.

Shapiro-Wilk Test: The Shapiro-Wilk test was used to assess the normality of the distribution of accuracy and F1-measure results for each method. This test is particularly appropriate for small sample sizes and helps determine whether parametric or non-parametric statistical methods should be applied. The Shapiro-Wilk test statistic W as calculated in equation (4.42) (González-Estrada & Cosmes, 2019):

$$W = \frac{(\sum_{i=1}^n a_i x_i)^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad (4.42)$$

where:

x_i are the ordered sample values (from smallest to largest).

$\bar{x}$ is the sample mean.

a_i are constants generated from the expected values of the order statistics of a standard normal distribution and the covariance matrix of those order statistics.

Friedman Test: The Friedman test, a non-parametric test, was used to compare the performance of multiple methods across different chunks. This test is particularly useful when the assumption of normality is violated. The Friedman test statistic χ_F^2 is calculated in equation (4.43) (J. Liu & Xu, 2022):

$$\chi_F^2 = \frac{12}{nk(k+1)} \left[\sum_{j=1}^k R_j^2 \right] - 3n(k+1) \quad (4.43)$$

where:

n is the number of subjects.

k is the number of treatments (groups).

R_j is the sum of ranks for the $j - th$ treatment.

If the calculated value of χ_F^2 is larger than the critical value from the Chi-Square distribution table for $k - 1$ degrees of freedom, we reject the null hypothesis.

A significance threshold of $\alpha = 0.05$ and $\alpha = 0.1$ will be used for analysis. The usage of two levels is considered in order to enable a degree of freedom for the tolerance in

judging the performance of the method. This approach allows for a more nuanced interpretation of the results, accommodating varying degrees of statistical stringency and providing a broader perspective on the performance evaluation.

4.4.3 Datasets

An issue with the existing works on XSS is that they are evaluated on one type of XSS data or one specific data representation. Some researchers evaluated their approaches solely on payloads data (Kuppa et al., 2022), (Akaishi et al., 2019), (Maurel et al., 2022), (Lee et al., 2022), (Abaimov & Bianchi, 2019), (Chaudhary et al., 2021); while other solely on JavaScript files (Alazab et al., 2022), (Y. Huang et al., 2021), (Rozi et al., 2022). Additionally, some researchers evaluate on numerical dataset extracted from the payload (Mokbal, Dan, Imran, Lin, et al., 2019), (Mokbal et al., 2021), (Y. Zhou & Wang, 2019), (Tariq et al., 2021). This limits scope of evaluation which makes the assessment incomplete and may not reflect the real-world XSS detection scenarios accurately. The numerical dataset is a dataset that its features have been extracted by using statistical feature extraction methods.

In contrast to previous researchers, our proposed framework addressed this limitation by employing two types of datasets for the evaluation, namely, raw textual data, i.e., JavaScript and payload, and numerical data extracted from the payload. By incorporating multiple data types and data representation, the proposed framework demonstrates the capability to detect the XSS attack across diverse scenarios.

Given that these datasets are derived from real-world data collected and released over several years, there are no explicit drift labels or markers available. Literature indicates that attackers continually refine their techniques and develop new evasion strategies, leading to the emergence of new variants of XSS attacks over time, as detailed in Chapter 2 (Section 2.3.3). Consequently, while no specific drift-checking mechanisms were employed during the data collection phase, the analysis presented in Chapter 5 will assess whether these datasets exhibit feature drift and evaluate its impact on model performance. This analysis will help validate the assumption that the datasets contain feature drifts, thereby confirming the effectiveness and adaptability of the proposed methods in addressing evolving XSS attacks.

4.4.3.1 Numerical Datasets

In this part, we came across four numerical XSS attacks datasets used in this research with a purpose of demonstrating the efficiency of our proposed DQN-MAFS approach when it comes to the low and high number of dimensions. Such datasets consist of numerical features that are extracted by using statistical-based feature extraction method, we summarize them in Table 4.5., and we give details of each of them in the following sub-sections.

Datasets of (D1-N66) and (D2-N167): The first numerical dataset, D1-N66, was published by (Mokbal, et al., 2019) and updated in (Mokbal et al., 2021). We used the updated version in this study as the second dataset, D2-1N67. These two datasets are considered the first comprehensive XSS-based numerical features dataset extracted from real XSS attacks. The benign samples in this dataset were created by crawling

the top 50,000 websites ranked by Alexa. In contrast, the malicious samples were collected by crawling raw XSS repositories such as XSSed (XSSed Archive, 2015) and Open Bug Bounty (Open Bug Bounty, n.d.). Initially, the author extracted only 66 numerical features. Then (Mokbal et al., 2021) extracted more comprehensive features, in which 167 numerical features were extracted. Both datasets have been available online recently on GitHub (Fawaz, 2022). Furthermore, these datasets consist of three types of features: HTML, JavaScript, and URL.

Dataset of (D3-N30): The third numerical dataset D3-N30 was created by (Zhou & Wang, 2019). This dataset was gathered from multiple GitHub sources and security forms and contained around 6503 normal records and 3497 XSS payloads. Then 30 script and URL context features were obtained using open-source ontology modelling tools (Protégé) based on only four characteristics. The features of this dataset have been extracted using domain expertise-based and statistical-based methods.

Dataset of (D4-N30): The fourth numerical dataset used in this study is D4-30 which was initially collected by (Y. Fang, Li, et al., 2018) as a raw dataset from different sources, namely, GitHub (Duoergun, 2017) and XSSed (XSSed Archive, 2015). Then the features were extracted directly from the XSS payload by (Tariq et al., 2021) based on (Zhou & Wang, 2019); here also, only 30 features were extracted using the JSoup parser library. The raw dataset is collected from GitHub (Duoergun, 2017) and XSSed (XSSed Archive, 2015). Both datasets D3-N30 and D4-N30 are available on (IramTariq, 2020).

Table 4.5: Existing numerical XSS benchmarking datasets

Dataset name	Author name	No. of samples	No. of features	Source of malicious samples	Source of benign samples	Collection method
D1-N66	(Mokbal, et al., 2019)	100,000 benign samples, and 38,569 malicious samples	66 features	XSSed and Open Bug Bounty	50,000 Top Alexa websites	Crawling and parsers
D2-N167	(Mokbal et al., 2021)		167 features			
D3-N30	(Zhou & Wang, 2019)	6503 normal records and 3497 XSS payloads	30 features	XSSPL and XSSed	GitHub	Ontology modeling tools (Protégé)
D4-N30	(Y. Fang, Li, et al., 2018)	135,507 normal records and 16,151 XSS payloads.	30 features	XSSed	GitHub	JSoup parser library

4.4.3.2 Raw Textual Datasets

To evaluate the performance of the proposed DA-XSS-AD framework and demonstrate the model capacity, the second model 2TS-DFS was evaluated by employing two of raw text XSS datasets are JavaScript files and payload samples as shown in Table 4.6. The significant difference between the XSS payload and JS file, is that the payload is just single line script code embedded within web pages or URL components. The payload is prepared by the attacker by using a combination of JS functions, HTML attributes, HTML methods, and events that are included inside HTML tags. While the JS file is multiple script code lines written only by using JS programming language.

JavaScript dataset (D5-JS) consists of 8000 unique JavaScript samples. In particular, they are composed of 4000 benign and 4000 malicious JavaScript files. These JavaScript files were obtained separately, all benign samples come from Alexa (Cooper, 2020) top 100 by depth crawling, and the malicious samples were from GitHub (HynekPetrak, 2017) and the malicious platform VX Heavens (VX Vault, 2017).

XSS payloads (D6-PL) dataset is collected from multiple GitHub sources and security, consisting of 10,000 records. This dataset consists of 6503 normal records and 3497 XSS payloads. The malicious records are collected from different forums (1) (İsmail Taşdelen, 2021), (2) (Duoergun, 2017), and (3) XSSed (XSSed Archive, 2015). While the benign samples are gathered from DMOZ database (The Directory of the Web, n.d.). Although they are different in trigger process, storage position and output position, their payloads have common characteristics, such as consisting of the same sensitive words and characters (Zhou & Wang, 2019).

Table 4.6: Existing raw text-based benchmarking datasets

Dataset name	No. of samples	Source of malicious samples	Source of benign samples	Collection method
D5-JS	4,000 benign samples, and 4000 malicious samples	(HynekPetrak, 2017), (VX Vault, 2017)	100 Top Alexa websites	Crawling and parsers
D6-PL	6503 normal records and 3497 XSS payloads	(İsmail Taşdelen, 2021), (Duoergun, 2017), (XSSed Archive, 2015)	DMOZ database	Crawling and parsers

4.4.4 Experimental Design

For experimental evaluation, we used Google Colab pro environment, a web-based platform that provides a Jupyter notebook environment to carry out our all experiments. Python, a popular programming language, was used to implement our experimental design, Colab uses Python 3.6.9. Google Colab pro+ provides access to GPU runtime that comes with Intel Xeon CPU @2.20 GHz, 13 GB RAM, Tesla K80 accelerator, and 12 GB GDDR5 VRAM, as presented in Table 4.7. The experimental design indicates to the setting of the data decomposition and parameters of the evaluated models for producing the evaluation results. We present the experimental design into two parts. The first one is dedicated to DQN-MAFS and the second is dedicated to 2TS-DFS.

Table 4.7: Hardware details of the implementation environment

Component	Specification
Environment	Google Colab pro
Python version	Python 3.6.9
GPU/RAM	Tesla K80 accelerator/12 GB GDDR5 VRAM
CPU/RAM	Intel Xeon CPU @2.20 GHz/13 GB RAM

In the following subsections, it is important to highlight that some of the DQN-MAFS and 2TS-DFS's hyperparameters were set to align with those used in selected benchmark studies, specifically Tariq et al. (2021), K. Liu et al. (2021), and Y. Huang et al. (2021), to ensure comparability of results. The remaining hyperparameters were fine-tuned through trial and error, guided by empirical testing to achieve the best possible balance between performance and computational efficiency. This approach allowed us to optimize the model's effectiveness while ensuring it remained consistent with established standards in the field.

Furthermore, to ensure the validity of our evaluation, we reproduced the same evaluation methodology that has been used by other researchers and established benchmarks. This approach includes using the same datasets and parameters as those utilized in previous studies, which allows for an objective comparison of our methods against the benchmark results. By aligning our evaluation process with these established practices, we mitigate potential threats to validity, ensuring that our findings are both reliable and comparable within the context of the field. This methodical replication strengthens the credibility of our results by providing a robust framework that has been validated by prior research.

4.4.4.1 DQN-MAFS

The experiments of the DQN-MAFS model were performed using four numerical XSS datasets, namely, D1-N66, D2-N167, D3-N30, and D4-N30. The two benchmarks used for comparison are Tariq et al., (2021) and K. Liu et al., (2021). We benchmarked our work with Tariq et al., (2021) because of their adaptive XSS detection method based on genetic algorithm and RL, while the method of K. Liu et al., (2021) because of their dynamic feature selection based on multi-agent RL approach. Both these two benchmarks are re-implemented in a similar design to our method to reproduce their results using four datasets, as mentioned earlier. For evaluation, each dataset was partitioned into chunks, similar to how data streams feed incremental learning models. The method was required to provide a different feature selection decision for each chunk. The total number of chunks is 9 for each dataset. The dataset splitting for training and testing depends on how many data samples we have. In this part, all used datasets are divided into 0.8 for training and 0.2 for testing as shown in Table 4.8. This

is due to the fact that we have divided each of the used datasets into chunks, in which the number of the data in each chunk is small; thus, we select the highest percentage of the data samples for training to generalize the model and avoid the under-fitting of the model. The chunks are provided to each of the agents simultaneously at the specific moment as well as to benchmarks to guarantee objective evaluation.

In our experiments, we set the DQN-MAFS with several parameters. Each agent has its own deep Q-network; the batch learning is set as shown in Equations (4.9 and 4.10). We use the ε -greedy policy to balance between the exploration and exploitation processes. The value ε -greedy policy is 0.97 and the ε decay value is 0.01 from 1 to 0 at the end of each episode. For the experience replay, the capacity of the replay buffer is set to 800. The Q-network is set in this approach as three neural network layers. ReLU is used as the activation function at the hidden layers 2nd and 3rd layers with 32 and 16 neurons, respectively. Whilst the loss function is mse, and the optimizer is Adam. The total number of epochs is 5, with a batch size of 32. The learning rate α is 0.001, and the discount factor γ is 0.995. The four predictors (SVM, KNN, DT, and LR) are set with different parameters. SVM is set with radial basis function (RBF) kernel, maximum iteration =8000, regularization proportion $c=0.2$, and probability = true. Also, five neighbors are used with the KNN. Furthermore, LR is set with a maximum iteration equal to 500. Finally, the DT is used with 15 estimators, and the random generator is initialized with 42. In the ACC and ALC methods, we used a sliding window with a size equal to 4 steps to calculate the reward contribution from the CM. The ALC method's threshold value is set at a 0.6 of accuracy rate. Table 4.9 summarize the aforementioned parameters of the DQN-MAFS model.

Table 4.8: Chunk composition and splitting

Name	Details
No. of chunks	Each dataset of four used numerical datasets is composed into 9 chunks
Splitting percentage	80% for training, and 20% for testing

Table 4.9: Parameters of the DQN-MAFS model

	Model	Details
RL parameters	Exploration and exploitation balance policy	ϵ –greedy policy with value of 0.97 and the ϵ decay value is 0.01 from 1 to 0
	experience replay	800
	discount factor γ	0.995
	learning rate α	0.001
DQN parameters	Network parameters	three neural network layers activation function is Relu 2 nd and 3 rd hidden layers with 32 and 16 neurons
	Training parameters	loss function is mse the optimizer is Adam 5 epochs batch size of 32 learning rate α is 0.001
	SVM	radial basis function (RBF) kernel maximum iteration =8000 regularization proportion $c=0.2$
	KNN	five neighbors
Classifiers	DT	15 estimators
	L.R	random generator is initialized with 42 maximum iteration equal to 500
FARD-DFS	Sliding window used with ACC and ALC for CM	Window size equal to 4 steps
	ALC threshold	0.6 of accuracy rate

4.4.4.2 2TS-DFS

The aim of this model is to detect the XSS attack from both perspectives of spatial and temporal dynamic feature selection. The experiments were evaluated using two raw text XSS datasets, namely, JS-based samples dataset (D5-JS), and payload-based samples (D6-PL).

The benchmarks used for comparison are (Y. Huang et al., 2021) and our DQN-MAFS models. We benchmarked proposed work with (Y. Huang et al., 2021) because of their

adaptable XSS context analysis method (JSContana) based on the word2vec model integrated with two layers of Bi-LSTM. Also, the evaluation was benchmarked with the most performance achieving models of DQN-MAFS because of proposing a temporal dynamic feature selection method based on multi-agent RL approach. Each method was proposed with its novel reward distribution method for distributing the reward value among agents based on their real contribution. These methods are namely, accumulative contribution (ACC), alternative contribution (ALC) and impurity based (IM), and one action at one time (OA-OT). In this experiment, we only selected the three methods that achieved height detection performance are ACC, ALC, and OA-OT methods. Finally, we also benchmarked the proposed method with a static word embedding method that is based on the Word2vec model for word embedding representation and TextCNN model for learning and classification tasks.

All the benchmarks are re-implemented in a similar design to our method to reproduce their results using two raw datasets, as mentioned earlier. For evaluation, each dataset was partitioned into chunks, similar to how data streams feed incremental learning models. The method was required to provide a different feature selection decision for each chunk. The total number of chunks are 4 and 5 for JS-based dataset and Payload-base dataset, respectively, because of the number of samples of these datasets is small in comparison with the numerical data sets' samples (that have been divided into 9 chunks) as explained in Section 4.4.4.1. In this part, both used datasets are divided into 0.8 for training and 0.2 for testing. This is due to the fact that we have divided each of the used datasets into several chunks in which the number of the data in each chunk is small; thus, we select the highest percentage of the data samples for training to generalize the model and avoid the under-fitting of the model.

In our experiments, we set the 2TS-DFS with several parameters as shown in Table 4.10. Firstly, the pre-processed JS and payload segments were trained with the Word2Vec model to obtain the static word embeddings representation. The parameters of Word2Vec are set with a vector size of 50, a window size of 8, a minimum count of 1, and vocabulary size of 2048. Secondly, we used a 2-layers BI-LSTM model to obtain a hidden state of 100 dimensions. The input data consisted of text sequences encoded as word embeddings, which were fed into the BI-LSTM model. During training, we used a learning rate of 0.001, a batch size of 100, train tokens of 400,000, and trained the model for 10 epochs on a dataset samples. The final output vector was obtained with dimension size of 100x2048. Thirdly, we applied the agglomerative clustering method to group the records into 35 clusters based on their spatial similarity. We used the Euclidean distance metric to measure the similarity between records and the Ward linkage method to merge clusters. Fourthly, we developed a Tabu search-based dynamic feature selection method to select the optimal feature set for each cluster within the same timestamp. The feature selection process used a move type of 2, aspiration criteria of 0.9 for both accuracy and F1-score, a tabu list length of 100, and a number of iterations equal to the multiplication of the length of Shape values of X_{train} by 10 = $(2^{10}) \cong 1000$. During testing, we assigned the testing records to similar training clusters by comparing their statistical information, which included minimum, maximum, mean, standard deviation, and median, with that of the training clusters. The statistical information of the testing records was compared with that of the training clusters using a nearest neighbor approach. Specifically, the testing records were assigned to the most similar training cluster based on the smallest Cosine similarity between their statistical information and that of the training clusters. Finally, we used a TextCNN algorithm for the classification task, which consisted of 4

convolutional layers with filter sizes of 7,15,25 and 35, respectively, tanh is used an activation function, and a max pooling layer for each convolutional layer, in dense layers relu and sigmoid are used as a activation function, respectively, for first and second dense layers, binary_crossentropy is used a loss function, optimizer is ‘adam’ in compiling the model. The model was trained using a learning rate of 0.001, a batch size of 30, and a total of 20 epochs with early stopping. Regularization was achieved using dropout with a rate of 0.2. The used datasets are divided into 0.8 for training and 0.2 for testing as presented in Table 4.11.

Table 4.8: Parameters of the 2TS-DFS model

	Model	Details
Word2Vec	Model network	vector size of 50 window size of 8 minimum count of 1 vocabulary size of 2048 hidden state of 100 dimensions learning rate of 0.001
Bi-LSTM	Model network	batch size of 100 train tokens of 400,000 Output vector dimension size of 100x2048
Agglomerative clustering	Metric to measure the similarity between records merge clusters method	Euclidean distance metric Ward linkage method
Tabu search	Tabu list size, and aspiration criteria Number of changes to generate new solution number of iterations	tabu list length of 100 aspiration criteria of 0.9 for both accuracy and F1-score 5 bits for each record move type of 2
TextCNN	TextCNN network	length of Shape values of X_train by 10 consisted of 4 convolutional layers with filter sizes of 7,15,25 and 35, respectively. max pooling layer for each convolutional layer tanh is used an activation function in dense layers relu and sigmoid are used as a activation function, respectively binary_crossentropy is used a loss function optimizer is ‘adam’ learning rate of 0.001
	Training parameters	batch size of 30 total of 20 epochs dropout with a rate of 0.2

Table 4.9: Chunk composition and splitting

Name	Details
No. of chunks	D5-JS is composed into 4 chunks while the D6-PL is composed into 5 chunks
No. of samples per chunk	2000 samples
Splitting percentage	80% for training, and 20% for testing
No. of samples for training per chunk	1600 samples
No. of samples for testing per chunk	400 samples

4.4.4.3 DA-XSS-AD

Given the dual processing modes inherent to the DA-XSS-AD framework, its parameters are dynamically selected based on the nature of the input data. Specifically, when the input data comprises, numerical features extracted from XSS payloads, the framework activates the DQN-MAFS mode for both learning and attack detection. Conversely, in scenarios where the input data is in the form of raw textual data, such as payloads or JavaScript files, the 2TS-DFS mode is triggered. This mode preprocesses the data through dynamic word embedding and utilizes the Tabu Search algorithm along with Agglomerative clustering for enhancing data representation and subsequent XSS attack detection.

The parameters governing DQN-MAFS mode encompass feature selection configurations, reinforcement learning hyperparameters, and the selection of machine learning classifiers. These parameters are chosen in alignment with the parameters outlined in Section 4.3.3.1, ensuring optimal adaptation to the nature of the data. In the case of 2TS-DFS mode, parameters involving dynamic word embedding, Agglomerative clustering, and Tabu Search parameters are set according to the parameters detailed in Section 4.3.3.2, to facilitate accurate data preprocessing and clustering-based detection.

To evaluate the performance of the DA-XSS-AD framework, a targeted assessment strategy has been devised. Specifically, we have selected 10 data chunks from various datasets to ensure comprehensive coverage of different data types. The datasets used include both numerical datasets (D1-N66, D2-N167, D3-N30, and D4-N30) and raw textual data (D5-JS and D6-PL) as detailed in Section 4.3.3. To create the sequence, we employed a random uniform distribution method for selecting the chunks. This approach was adopted to maintain an unbiased and balanced representation of the diverse dataset types. The resulting sequence of selected chunks is denoted as $Seq = \{D5 - JS - Ch0, D6 - PL - Ch1, D3 - 30 - Ch2, \dots, D4 - N30 - Ch9\}$. This sequence demonstrates the capability of the DA-XSS-AD framework to handle a variety of data types effectively. Accordingly, the DQN-MAFS and 2TS-DFS methods are activated based on the specific data type of the input, showcasing the framework's adaptability and robustness in managing different types of XSS data. Furthermore, the core focus of the performance assessment revolves around following metrics: classification accuracy, F1-measure and feature reduction rate to compare with the methods of Tariq et al., (2021), K. Liu et al., (2021), and (Y. Huang et al., 2021).

4.5 Chapter Summary

This chapter presented the methodology design of the fundamental research. Overall proposed DA-XSS-AD framework of the XSS attack detection was introduced to handle the data evolving. In addition, it presented the main components of the proposed framework including DQN-based multi-agent features selection (DQN-MAFS), and Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS). The main aim of the first model (DQN-MAFS) is also identified to provide a

temporal dynamic feature selection in the domain XSS detection by considering incremental learning and enabling knowledge updates for both feature selection and classification at the same time. The details of the DQN-MAFS model and its sub-components have been demonstrated in this chapter, starting from problem formulation, general methodology, preparation of data streaming, DQN-MAFS algorithm, training of DQN-MAFS algorithm, and detector. On the other hand, this chapter presented another proposed model, called 2TS-DFS (Tabu Search based Tempo-spatial Dynamic Feature Selection), in which it was designed for JavaScript (JS) or payload data and utilizes text-based preprocessing techniques. The 2TS-DFS model was proposed by combining both spatial and temporal feature selection in the XSS detection to enable the capability of selecting optimal features in both spatial and temporal dimensions. Several sub-components of the 2TS-DFS model, namely, data decomposition, pre-processing, adaptable context analysis, dynamic feature selection and classification, were illustrated in this chapter. In the next chapter, we present the evaluation results and the analysis of them with the comparison with the benchmarks.

CHAPTER 5

RESULTS AND DISCUSSION

5.1 Introduction

In this chapter, the results and discussion of the proposed dynamic and adaptive XSS attack detection framework (DA-XSS-AD) and its sub-models, are presented. The chapter is organized into several sections and sub-sections to provide a comprehensive analysis of the performance and effectiveness of the framework, as well as the individual sub-models as depicted in Figure 3.2, within Chapter 3 (Section 3.4). We begin by section 5.2 that presents the results of the DQN-MAFS model, which focuses on data chunks and overall analysis, examining its ability to capture temporal dynamic feature selection and maintain accurate predictions. After that, section 5.3 provides a delve into the results of the 2TS-DFS model, which emphasizes temporal and spatial feature selection, and evaluating the feature reduction results of tabu search algorithm over iterations and over chunks. Following the analysis of the sub-models, section 5.4 presents the results of the DA-XSS-AD framework as a whole, assessing its accuracy, F1-scores, and feature reduction rates across various data types and representations. Throughout the chapter, we engage in detailed discussions to interpret the results, highlighting strengths, and comparing our approach with existing methods. Section 5.5 provides an overall discussion and conclusion, summarizing key findings and implications of the proposed models and DA-XSS-AD framework in enhancing web application security against XSS attacks. Finally, the overall discussion is followed by a chapter summary in Section 5.6.

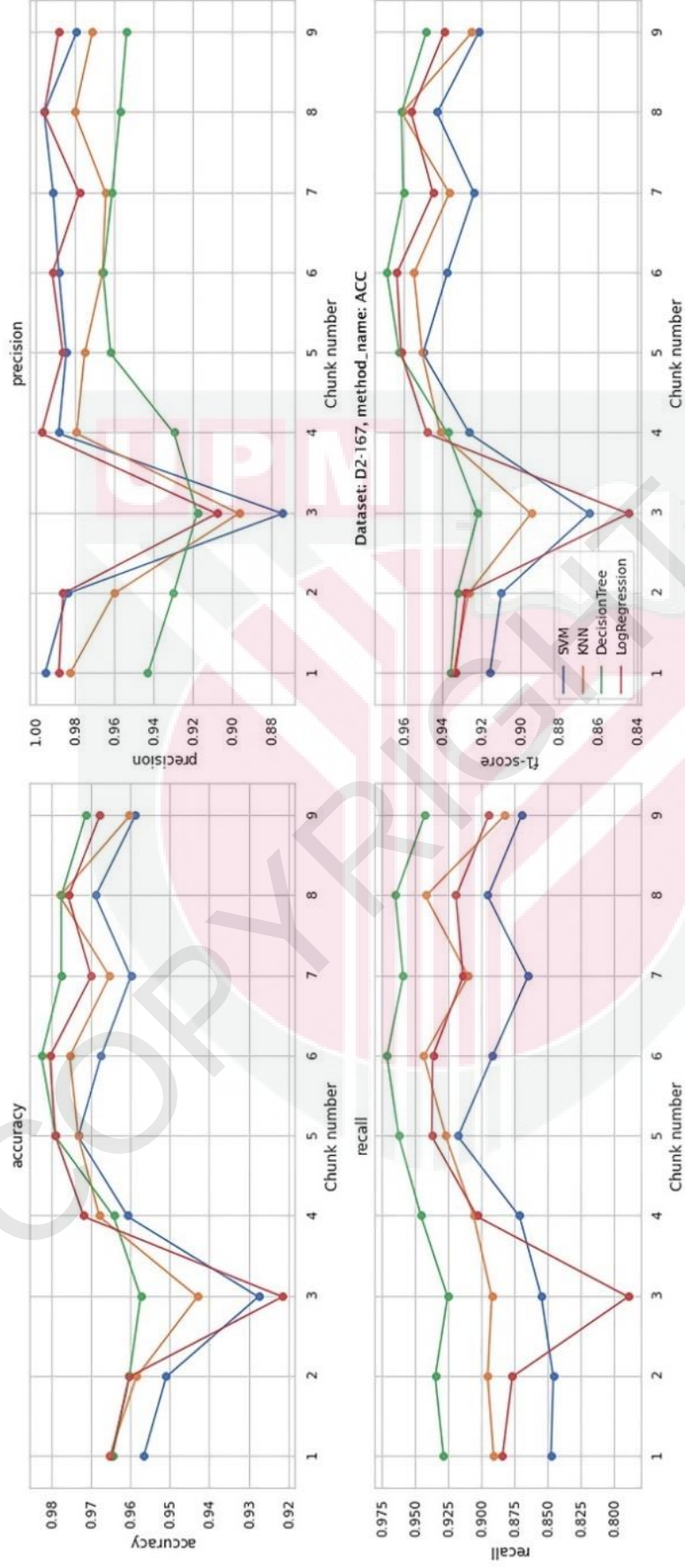
5.2 DQN-MAFS Model

The DQN-MAFS model was proposed to dynamically select features for numerical representation over time and to handle feature drift caused by the evolving nature of XSS attacks. This section presents the evaluation results of the DQN-MAFS model and its four different FARD-DFS approaches on four numerical datasets with varying dimension sizes. The evaluation begins by showcasing the performance of DQN-MAFS model across using different ML classifiers, to show the consistency of the performance and select the best classifiers that maintain highest prediction accuracy. Subsequently, the overall evaluation results of the proposed model, are introduced, including the mean, maximum (max), and standard deviation (std) of the classification metrics. These findings shed light on the effectiveness of the DQN-MAFS sub-model in dynamically adapting to changing feature patterns and its ability to maintain high prediction accuracy in detecting XSS attacks over time.

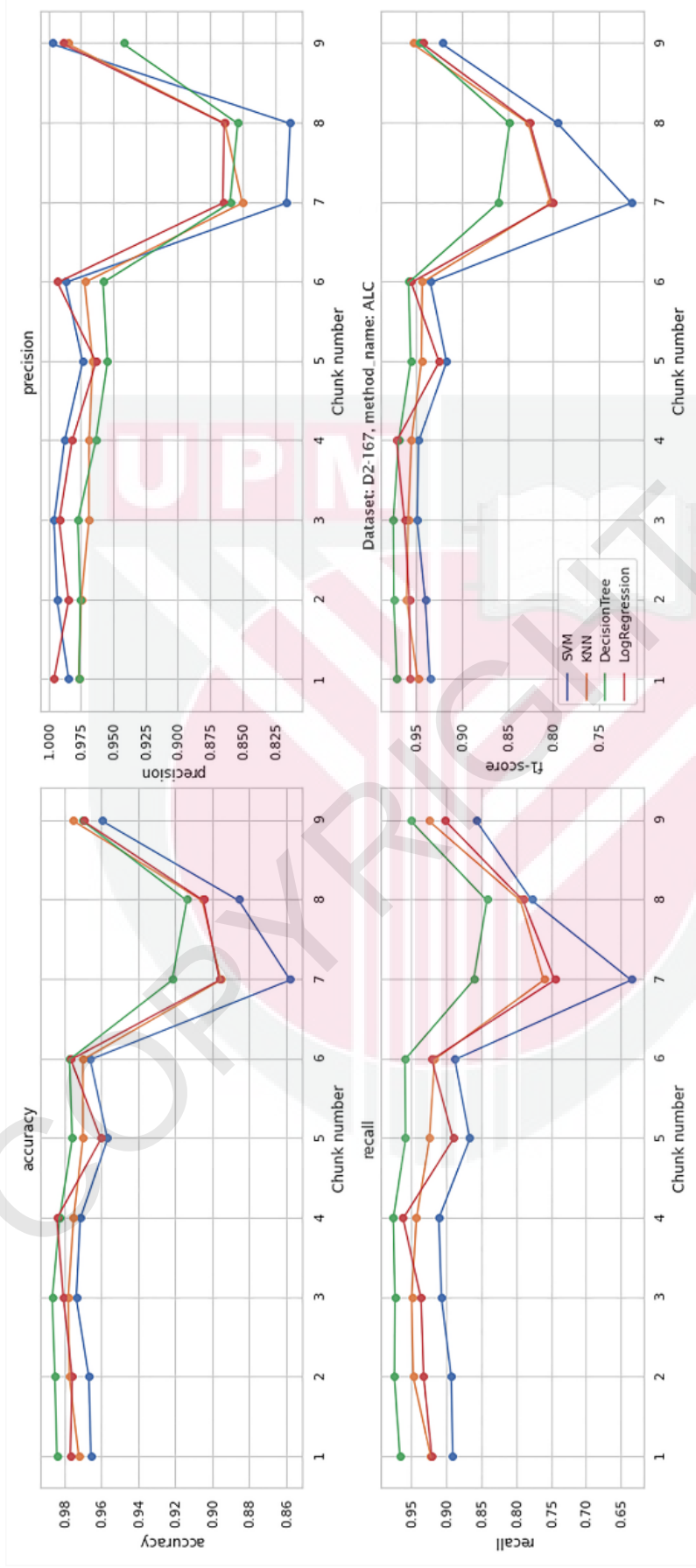
5.2.1 Evaluation of DQN-MAFS With Different Classifiers

The first set of DQN-MAFS results includes accuracy, precision, recall and F1-measure evaluation for datasets across multiple data chunks. The latter considers both recall and precision by measuring their harmonic mean; Thus, the discussion will focus on the classification accuracy and F1-measure within this section. In order to evaluate our proposed DQN-MAFS method of temporal dynamic feature selection for evolved XSS detection, four classifiers were used, namely, SVM, KNN, DT, and LR. Thus, we select the largest dataset in terms of features size, i.e., D2-N167, and provide its evaluation over data chunks in Figure 5.1, the method of data chunking is illustrated

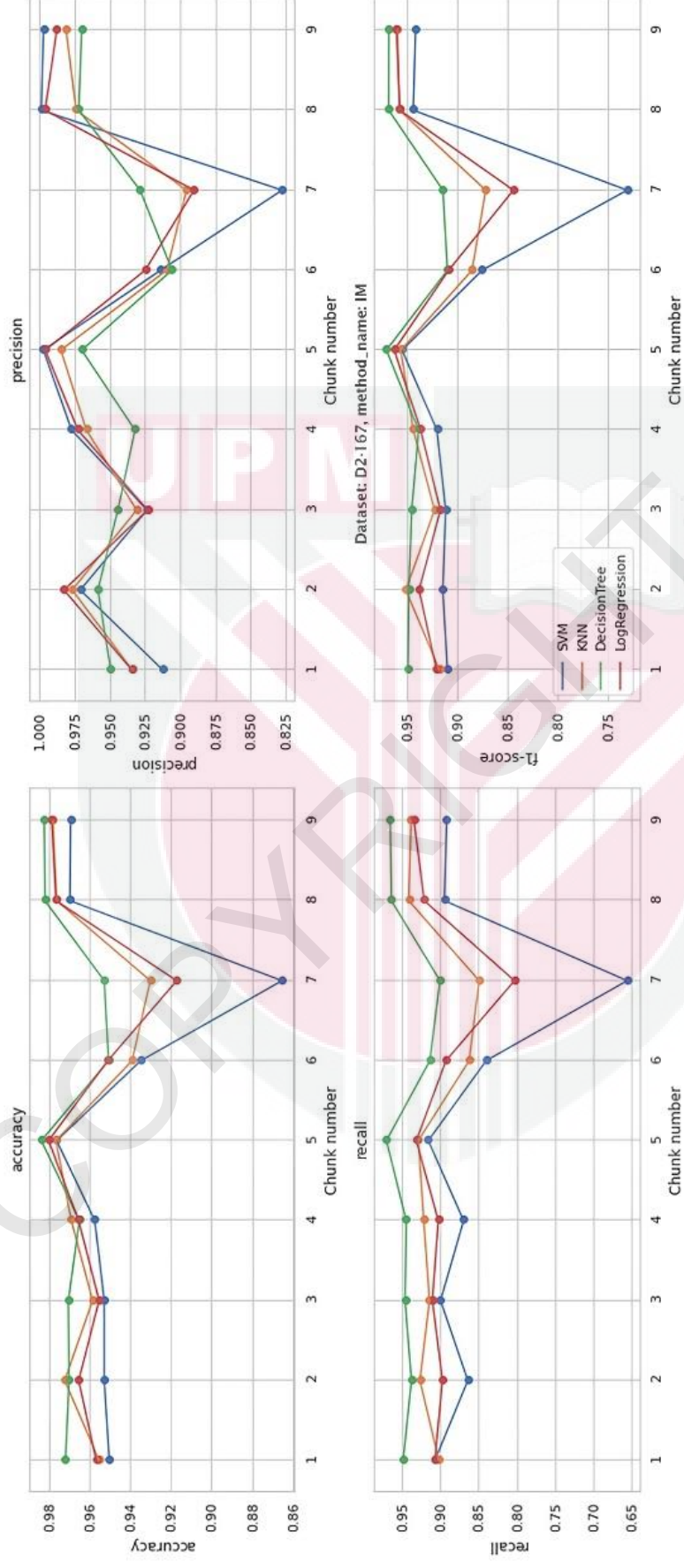
in Chapter 3 (Section 4.2.1). It is observed that each method has a variable classification accuracy and F1-measure according to the chunk. However, in all methods, the minimum and maximum values were in the range of 85.79% - 98.81% and 71.45% - 97.84% for accuracy and F1-measure, respectively. Another observation is that a drop in the accuracy and F1 measure to the values 85.79% and 71.45%, 86.51% and 73.09%, respectively, has occurred in chunk 7 with IM and ALC methods as shown in Figure 5.1 (b) and (c). Such a drop is arising due to the methods' reliance on data characteristic and complexity of data distribution could pose challenges for them, leading to the observed accuracy drops. On the other side, another drop has shown in the accuracy and F1 measure to 92.14% and 84.45%, respectively, within the ACC method in chunk 3 as depicted in Figure 5.1 (a). This drop could be attributed to the specific characteristics of data in chunk 3 might deviate from the model assumptions, leading to suboptimal feature selections and, consequently, reduced performance. Furthermore, OA-OT approach has less steady performance compared with other methods due to its sequential nature of their decision changes that renders it responsive to shifts in data pattern and distributions. Overall, all methods have provided an outstanding performance with four classifiers used despite the variation. This is interpreted by the fact that each rewarding method focuses on one aspect that can be captured in a specific chunk and lost partially in others.



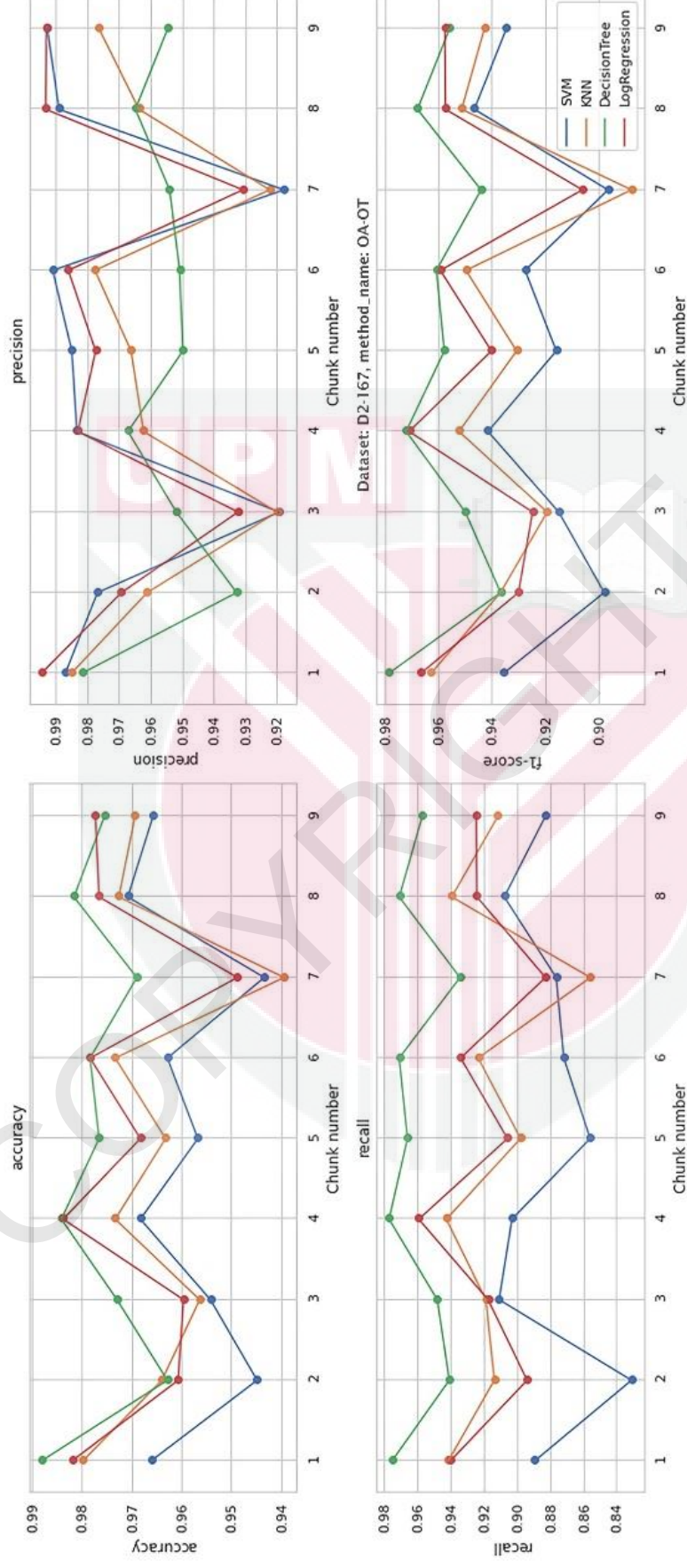
(A) ACC



(B) ALC



(C) IM



(D) OA-OT

Figure 5.1: Evaluation of DQN-MAFS and its FARD-DFS approaches with different classifiers in terms of accuracy, precision, recall, and f1-measure metrics, by using D2-N167 datasets

In this experiment, we have used both accuracy and F1-score as our primary performance metrics (selecting criteria). Although these classifiers showed similar overall performance in terms of accuracy and F1-score, we observed that the Decision Tree (DT) consistently performed slightly better in specific data chunks, as depicted in Figure 5.1. Particularly, DT demonstrated superior performance in chunks 7 and 8 with the FARD-DFS-IM, FARD-DFS-ACC, and FARD-DFS-OA-OT methods, as well as in chunks 2 and 3 with the FARD-DFS-ALC and FARD-DFS-OA-OT methods. In contrast, SVM, KNN, and LR exhibited a decline in accuracy and F1-score in these particular chunks. DT classifier produced the highest accuracy (98.81%), F1-measure (97.84%), and area under the ROC curve (AUC, 98.41%) with OA-OT method, indicating that DT has a superior balance of precision and recall, and excellent classification performance. Given its consistent performance and stability across various data chunks, DT was selected as the base classifier for the proposed DQN-DFS model.

5.2.2 Comparative Analysis

In this section, a comparison of the proposed DQN-MAFS method with two studies Tariq et al., (2021) that have proposed an adaptive method to detect the evolved XSS attacks; this method was evaluated using two datasets, D3-N30 and D4-N30 datasets. The second work is K. Liu et al., (2021), which proposed a method for dynamic feature selection based on MARL. This comparison is conducted using four numerical datasets with various feature sizes in each dataset. The goal of using different sizes and numbers of features for the data is to explore the performance variations according to these factors. According to the previous results, we selected the DT classifier as a

predictor with our proposed approach because of its results' superiority over other used classifiers such as SVM, KNN, and LR. In the subsequent sub-sections, we present the analysis of the results for the mentioned datasets.

In our results analysis, we prioritized the metrics of accuracy and F1-score over recall and precision due to several critical considerations. Accuracy provides a comprehensive measure of the model's ability to predict both positive and negative samples correctly, offering a broad and easily interpretable indication of general performance. The F1-score, as a harmonic mean of precision and recall, captures the balance between these two metrics, particularly in scenarios with imbalanced datasets. This balance is essential because accuracy alone might not be sufficient in such contexts, where the model could achieve high accuracy by predominantly predicting the majority class and neglecting the minority class. The F1-score addresses this by penalizing models that fail to balance precision and recall, thereby offering a more nuanced evaluation of performance.

We emphasize higher mean accuracy and F1-score to ensure the model's overall reliability and effectiveness in real-world applications. A higher mean accuracy indicates robust performance across the entire dataset, while a higher mean F1-score signifies that the model maintains a good balance between precision and recall, effectively identifying positive cases without producing excessive false positives. Additionally, we consider the standard deviation (Std) of these metrics to assess the stability and consistency of the model's performance across different data chunks. A lower standard deviation indicates greater reliability, and evaluating the maximum (Max) values highlights the model's peak performance, showcasing its highest

achievable effectiveness. This approach ensures that we assess the model's general effectiveness, stability, and potential peak performance, thereby confidently demonstrating the robustness and reliability of our proposed method in various scenarios, ensuring its practical utility and effectiveness.

5.2.2.1 D1-N66 Dataset

Another evaluation was performed on 66 numerical features dataset is provided in Table 5.1. As it is shown, we find that the best achieved mean and maximum accuracy was for FARD-DFS-IM with values of 99.59% and 100%, respectively. Furthermore, we find that the mean and maximum achieved F1-score were for FARD-DFS OA-OT and FARD-DFS-IM, with values of 90.75% and 100%, respectively. Furthermore, we observe that the accuracy of all FARD-DFS variants is always higher than 99%. Hence, we readily conclude that FARD-DFS is superior. In addition, we observe that FARD-DFS-IM has provided the least standard deviation (Std) in accuracy 0.44%, which is an indicator of more expected performance.

Table 5.1: Numerical comparison of our developed FARD-DFS with Tariq et al., (2021) and K. Liu et al., (2021) by using D1-N66 dataset

Methods Name	Accuracy			Precision			Recall			F1-measure		
	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std
FARD-DFS-ACC	99.42%	99.82%	0.57%	86.04%	96.77%	16.89%	85.21%	97.22%	22.45%	85.27%	97.00%	20.23%
FARD-DFS-ALC	99.06%	99.82%	1.13%	85.03%	95.00%	12.09%	81.36%	95.00%	16.70%	82.98%	94.74%	14.32%
FARD-DFS-IM	99.59%	100%	0.44%	89.91%	100%	15.40%	88.50%	100%	9.62%	88.62%	100%	11.31%
FARD-DFS OA-OT	99.52%	99.91%	0.45%	89.79%	100%	7.59%	91.90%	100%	7.30%	90.75%	97.44%	6.79%
K. Liu et al., (2021)	94.88%	98.30%	2.75%	96.58%	99.85%	3.91%	84.69%	93.58%	6.67%	90.20%	96.61%	5.44%
Tariq et al., (2021)	87.49%	94.80%	9.57%	84.58%	94.40%	19.02%	90.44%	97.86%	7.33%	85.94%	94.63%	14.09%

5.2.2.2 D2-N167 Dataset

The second used dataset has a high number of features, which is 167. Hence, the evaluation based on this dataset tests the model's capability to perform on a large number of extracted features. Table 5.2 shows that the most superior method in terms of accuracy and F1-score is FARD-DFS OA-OT. The reached mean and maximum accuracy rates were 97.66% and 98.81%, respectively and the reached mean and maximum F1-score were 95.81% and 97.84%, respectively. This shows an overall outperformance of FARD-DFS OA-OT over the remaining models.

Furthermore, it is observed that all variants of FARD have provided good performance compared with other models. For example, the maximum accuracy was always higher than 98% for all FARD variants. This provides the capability of FARD to attain needed dynamic knowledge for selecting the most relevant features compared with other models. Furthermore, The least accuracy and F-measure have been observed at Tariq et al., (2021) with a value of 87.19% and 88.93%.

In order to capture the consistency in the performance, we calculated the standard deviation. The lower values of the latter indicate to more expected performance. It is observed that the least std has occurred in OA-AT for accuracy, which was 0.77% compared with the std of Tariq et al., (2021) 13.21%.

Table 5.2: Numerical comparison of our developed FARD-DFS with Tariq et al., (2021) and K. Liu et al., (2021) by using D2-N167 dataset

Methods Name	Accuracy			Precision			Recall			F1-measure		
	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std
FARD-DFS-ACC	97.06%	98.27%	0.92%	94.68%	96.58%	1.73%	94.80%	97.09%	1.67%	94.78%	96.83%	1.63%
FARD-DFS-ALC	96.64%	98.67%	2.81%	94.00%	97.72%	4.86%	93.99%	97.57%	5.12%	93.99%	97.53%	4.98%
FARD-DFS-IM	96.99%	98.38%	1.22%	94.77%	97.19%	2.26%	94.39%	97.12%	2.37%	94.58%	97.05%	2.23%
FARD-DFS OA-OT	97.66%	98.81%	0.77%	95.63%	98.16%	1.37%	96.00%	97.70%	1.57%	95.81%	97.84%	1.34%
K. Liu et al., (2021)	95.56%	97.01%	0.84%	99.36%	99.85%	0.39%	84.67%	89.57%	2.83%	91.41%	94.30%	1.67%
Tariq et al., (2021)	87.19%	95.38%	13.21%	91.99%	95.93%	1.86%	87.80%	99.54%	16.55%	88.93%	95.20%	9.38%

5.2.2.3 D3-N30 Dataset

The analysis of obtained results of our model and the benchmarks based on the D3-N30 dataset is detailed in Table 5.3. It can be found that the best-achieved accuracy, a mean of 98.28%, and a maximum value of 100% were for FARD-DFS-IM and K. Liu et al., (2021), respectively. Furthermore, the best-achieved F1-measure in terms of mean was for FARD-DFS-ACC and in terms of maximum value was for K. Liu et al., (2021) for values of 97.38% and 100%, respectively. This implies that FARD-DFS has better performance overall than the benchmarks regarding accuracy and F-measure. Another observation is that Tariq et al., (2021) have a recall for both mean and maximum values of 99.97% and 100%, respectively. However, FARD-FDS is better at balancing true positive and negatively predicted results, which means more reduction of false alarms and attaining good optimistic predictions. Lastly, we observe that FARD-DFS-ACC has achieved the least std, which is 0.51%. Again, this is an indicator of a better estimation of the performance of the approach.

Table 5.3: Numerical comparison of our developed FARD-DFS with Tariq et al., (2021) and K. Liu et al., (2021) by using D3-N30 dataset

Methods Name	Accuracy			Precision			Recall			F1-measure		
	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std
FARD-DFS-ACC	98.28%	99.00%	0.51%	98.53%	100%	1.64%	96.29%	98.65%	1.82%	97.38%	98.67%	0.93%
FARD-DFS-ALC	98.00%	99.50%	1.12%	98.95%	100%	2.08%	95.13%	98.59%	2.32%	96.98%	99.29%	1.69%
FARD-DFS-IM	97.50%	99.00%	1.48%	98.36%	100%	1.89%	94.32%	97.33%	2.66%	96.27%	98.65%	2.03%
FARD-DFS OA-OT	97.83%	99.50%	3.72%	99.77%	100%	0.68%	93.90%	98.64%	9.97%	96.51%	99.32%	6.24%
K. Liu et al., (2021)	97.00%	100%	1.41%	98.32%	100%	3.33%	92.59%	100%	3.14%	95.32%	100%	2.32%
Tariq et al., (2021)	93.48%	98.57%	5.77%	86.98%	97.25%	11.55%	99.97%	100%	0.05%	92.63%	98.55%	7.16%

5.2.2.4 D4-N30 Dataset

The results of comparing FARD-DFS with other methods for the D4-N30 dataset are presented in Table 5.4. It is found that FARD-DFS-IM and FARD-DFS-ACC obtain the best mean and maximum accuracy with values of 98.37% and 99.01%, respectively. Similarly, the best mean and maximum achieved F1-score are 98.42% and 99.10% for FARD-DFS-IM and FARD-DFS-ACC, respectively. K. Liu et al., (2021) has provided a higher precision; however, the recall was relatively low 95.63% for mean. On the hand, Tariq et al., (2021) has accomplished a higher mean recall of 99.99%; however, the precision was relatively low, 93.32%. This shows that FARD-DFS was generally superior to both K. Liu et al., (2021) and Tariq et al., (2021) from two perspectives of positive and negative predictions. Lastly, the minimum std for accuracy is observed at FARD-DFS-ACC, which is 0.48%. This is an indicator of more expected performance. Another observation that can be made from this dataset results and the previous ones is that the accuracy was higher than recall in general. This is a common observation when the dataset contains more negative samples than positive ones. This leads to higher accuracy than recall because the model tends to predict negative samples which is not captured by the recall.

Table 5.4: Numerical comparison of our developed FARD-DFS approaches with Tariq et al., (2021) and K. Liu et al., (2021) by using D4-N30 dataset

Methods Name	Accuracy			Precision			Recall			F1-measure		
	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std	Mean	Max	Std
FARD-DFS-ACC	98.29%	99.01%	0.48%	99.52%	100%	0.47%	97.20%	98.36%	0.83%	98.35%	99.10%	0.47%
FARD-DFS-ALC	96.39%	98.74%	2.97%	99.13%	100%	0.91%	93.95%	97.86%	5.71%	96.38%	98.85%	3.18%
FARD-DFS-IM	98.37%	98.98%	0.49%	99.57%	100%	0.60%	97.31%	98.29%	0.63%	98.42%	99.06%	0.47%
FARD-DFS OA-OT	97.98%	99.06%	0.84%	99.43%	100%	0.69%	96.70%	98.39%	1.74%	98.04%	99.09%	0.86%
K. Liu et al., (2021)	97.36%	98.27%	0.58%	99.32%	100%	0.73%	95.63%	98.00%	1.49%	97.43%	98.42%	0.57%
Tariq et al., (2021)	96.66%	97.92%	0.82%	93.32%	95.85%	1.64%	99.99%	100%	0.01%	96.53%	97.87%	0.88%

5.2.3 Discussion

We present a comparative analysis of the mean and max accuracy for all methods in Table 5.5. The results provide that the majority of values of FARD-DFS are superior over the benchmarks K. Liu et al., (2021) and Tariq et al., (2021). Another observation is that FARD-DFS OA-OT is more superior when the number of features is high compared with FARD-DFS-ACC, which has behaved better for a low number of features around 30, while FARD-DFS-IM has behaved better with a medium number of features around 66. Based on the data presented in Tables 5.1- 5.5, we observed that the highest mean classification accuracy of 99.59% was achieved by the FARD-DFS-IM method using the D1-N66 dataset. Similarly, the FARD-DFS-IM method also attained the highest mean F1-score of 98.42% using the D4-30 dataset. In addition, we find that both K. Liu et al., (2021) and Tariq et al., (2021) have a degradation in the performance when the number of features increases, which means less scalability. This can be interpreted by the powerfulness of the reward calculation and distribution in FARD-DFS. Furthermore, the dynamicity handling of FARD-DFS is consistent and promising when it is used against the evolving nature of XSS attacks in an incremental or stream learning manner.

An additional observation derived from the analysis of the methods' results with using different datasets, is that the accuracy generally exceeded the recall and precision, particularly with the D1-N66 dataset. This phenomenon is notably observed when the dataset contains a greater number of negative samples compared to positive ones as shown in Section (4.4.2.1). As the D1-N66 consists of 100,000 benign samples, and 38,569 malicious samples. The prevalence of negative samples biases the model's

predictions toward negative classifications, thereby increasing the accuracy metric. However, this does not necessarily reflect the model's ability to correctly identify positive samples, which is more accurately captured by the recall metric.

In contrast, when analyzing the results from the D2-N167, D3-N30, and D4-N30 datasets, we found that the recall was slightly lower than both accuracy and precision. This slight underperformance in recall compared to other metrics further underscores a common challenge in handling imbalanced datasets, where the model's performance on the minority class (positive samples) may be suboptimal despite seemingly high overall accuracy.

Actually, in the analysis of the results, we based on the accuracy and F1-score metrics, because the latter is a harmonic mean of the recall and precision metrics. F1-Score serves as a harmonic mean of precision and recall, capturing the balance between these two metrics. It is particularly useful in situations where the dataset is imbalanced, as it considers both false positives and false negatives. By combining precision and recall into a single metric, the F1-score offers a more nuanced view of the model's effectiveness in handling positive cases.

Table 5.5: Overall accuracy results of the methods with respect to different feature size datasets

Methods Name	D1-N66 dataset		D2-N167 dataset		D3-N30 dataset		D4-N30 dataset	
	Mean	Max	Mean	Max	Mean	Max	Mean	Max
FARD-DFS-ACC	99.42%	99.82%	97.06%	98.27%	98.28%	99.00%	98.29%	99.01%
FARD-DFS-ALC	99.06%	99.82%	96.64%	98.67%	98.00%	99.50%	96.39%	98.74%
FARD-DFS-IM	99.59%	100%	96.99%	98.38%	97.50%	99.00%	98.37%	98.98%
FARD-DFS OA-OT	99.52%	99.91%	97.66%	98.81%	97.83%	99.50%	97.98%	99.06%
K. Liu et al., (2021)	94.88%	98.30%	95.56%	97.01%	97.00%	100%	97.36%	98.27%
Tariq et al., (2021)	87.49%	94.80%	87.19%	95.38%	93.48%	98.57%	96.66%	97.92%

As an assessment step for demonstrating the statistical significance of our proposed FARD-DFS approaches' results compared to benchmark methods, we conducted some statistical tests. The primary focus was to establish the superiority of our approaches over the method proposed by K. Liu et al., (2021) and Tariq et al., (2021) concerning accuracy and F1-measure on the D2-167 dataset. These comparisons were conducted across all nine chunks, with the results presented in Table 5.6.

To ensure the validity of our statistical analysis, we initially conducted the Shapiro-Wilk test to assess the normality of the accuracy data distribution. The results yielded p-values of 0.3944, 0.0037, 0.2524, 0.9999, 0.9115, and 0.002036 for FARD-DFS-ACC, FARD-DFS-ALC, FARD-DFS-IM, FARD-DFS-OA-OT, K. Liu et al. (2021), and Tariq et al. (2021), respectively. These results indicate that the ALC variant and the Tariq et al. (2021) method exhibit statistically significant deviations from normality ($p < 0.05$). Given these deviations, it was important to complement our analysis with a non-parametric approach. Therefore, we conducted the Friedman test to account for the non-normal distribution of some data sets and to further validate the robustness of our findings. The Friedman test, which does not rely on the assumption of normality, confirmed the statistical significance of our proposed methods in comparison to the benchmark methods.

Conversely, the remaining methods (FARD-DFS-ACC, FARD-DFS-IM, FARD-DFS-OA-OT, and K. Liu et al. (2021)) did not significantly deviate from normality, as indicated by their p-values ($p > 0.05$). Therefore, we proceeded to conduct the t-test for these methods, which is appropriate under the assumption of normality. This combined approach allows us to provide a comprehensive analysis, ensuring that our

results are both statistically sound and robust across different methodological considerations.

Following the Shapiro-Wilk test, which indicated that the most of FARD-DFS methods did not significantly deviate from normality ($p > 0.05$), we proceeded with a t-test to evaluate the statistical significance of differences in accuracy and F1-measure between these methods and the benchmark methods. The t-test is particularly suited for this analysis, as it assumes normality of the data, allowing us to rigorously assess whether the observed performance improvements of our proposed methods are statistically significant when compared to the established benchmarks. The t-test was conducted under two significance levels, $\alpha=0.05$ and $\alpha=0.10$, to provide a nuanced understanding of our methods' performance. This dual-threshold approach allows us to distinguish between strong effects ($\alpha=0.05$) and more subtle, yet potentially meaningful, differences ($\alpha=0.10$). Observing the Table 5.7, we find that a statistical significance with a confidence level of more than 90% has been achieved to support the superiority of our proposed approaches over both K. Liu et al., (2021) and Tariq et al., (2021) for almost all variants of our approach regarding both accuracy and F1-measure. In another observation, we found that all variants were superior to Tariq et al., (2021) with respect to accuracy with confidence levels of 0.0518, 0.0639, 0.0811, and 0.0445 of p-values for ACC, ALC, IM, and OA-OT, respectively. Furthermore, we find that each of ACC, IM, and OA-OT were superior over K. Liu et al., (2021) for both accuracy and F-measure with p-values of 0.0152, 0.0238 and 0.0001, respectively; and 0.0075, 0.0105, 0.0726 and 0.0001 for F1-measure of all suggested variants respectively.

Finally, in case the results are not normally distributed, we applied the Friedman test as a non-parametric alternative to the t-test. This test is particularly useful as it ranks the methods rather than relying solely on averages, which can sometimes obscure meaningful differences. The Friedman test results yielded a Chi-square statistic $X_j^2 = 21.89$ with $p < .0001$, indicating that our proposed FARD-DFS methods significantly outperform the benchmark methods in terms of accuracy. This thorough statistical analysis underscores the efficacy of our proposed methods and their potential for application in real-world scenarios, particularly in environments where data characteristics may vary or evolve over time.

Table 5.6: Numerical comparison at data chunk level of our developed FARD-DFS with Tariq et al., (2021) and K. Liu et al., (2021) by using D2-N167 dataset

Chunk	Method Name	Accuracy	f1-score	Method Name	Accuracy	f1-score
1	ACC	96.46%	93.56%	IM	97.19%	94.92%
2		96.03%	93.23%		97.01%	94.82%
3		95.74%	92.16%		97.04%	94.54%
4		96.43%	93.73%		96.50%	93.85%
5		97.91%	96.20%		98.38%	97.05%
6		98.27%	96.83%		95.06%	90.98%
7		97.76%	96.00%		95.27%	91.43%
8		97.76%	96.09%		98.20%	96.82%
9		97.15%	94.83%		98.23%	96.80%
1	OA-OT	98.81%	97.84%	ALC	98.41%	97.12%
2		96.28%	93.67%		98.52%	97.47%
3		97.29%	94.99%		98.67%	97.53%
4		98.41%	97.20%		98.27%	96.95%
5		97.66%	95.78%		97.58%	95.62%
6		97.84%	96.07%		97.73%	95.84%
7		96.90%	94.40%		92.17%	86.03%
8		98.16%	96.78%		91.38%	84.77%
9		97.55%	95.58%		97.01%	94.62%
1	K. Liu et al., (2021)	97.01%	94.30%	Tariq et al., (2021)	94.20%	94.02%
2		95.45%	91.61%		95.13%	94.93%
3		96.21%	92.55%		70.92%	75.98%
4		95.53%	91.41%		88.42%	88.76%
5		95.78%	91.77%		93.80%	93.63%
6		96.14%	92.42%		93.56%	93.59%
7		94.48%	89.09%		95.38%	95.20%
8		94.41%	89.26%		94.62%	94.34%
9		95.06%	90.25%		58.71%	69.91%

Table 5.7: t-test statistical significance testing of the accuracy and F1-measure results in comparison with benchmarks methods

Methods Name	t-test			
	Accuracy		F1-measure	
	K. Liu et al., (2021)	Tariq et al., (2021)	K. Liu et al., (2021)	Tariq et al., (2021)
FARD-DFS-ACC	0.0152	0.0518	0.0075	0.0847
FARD-DFS-ALC	0.0238	0.0639	0.0105	0.1378
FARD-DFS-IM	0.1806	0.0811	0.0726	0.2335
FARD-DFS OA-OT	0.0001	0.0445	0.0001	0.0581

5.3 2TS-DFS Model

The second sub model was proposed to select features spatially and temporally using hidden state representations to detect the evolving nature of XSS attacks. The hidden states capture dynamic word embedding representation generated by a word2vec model with two layers of Bi-LSTM models, as described in the Chapter 3 (Section 4.3.3). This section presents the evaluation results of the 2TS-DFS model on two raw textual datasets, namely payload and JS data. The evaluation focuses on the model's performance of training and testing, as well as comparisons with state-of-the-art methods over different data chunks. Furthermore, the evaluation results also present the memory enhancement achieved by the model to show the efficiency of the temporal and spatial feature selection in reducing feature dimensions. Additionally, the overall discussion of the proposed model is introduced, including the mean, minimum (min), maximum (max), and standard deviation (std) of the classification metrics. These findings reveal the effectiveness of the 2TS-DFS sub-model in dynamically selecting temporal spatial features to adapt to changing feature patterns of the XSS attack.

5.3.1 Evaluation of 2TS-DFS Model

This section offers an in-depth analysis of the training and testing evaluations of 2TS-DFS over different data chunks, aiming to showcase the search performance of the tabu search algorithm. Given that a single solution must be chosen from the array of solutions derived through tabu search, our selection process involves identifying the solution that has yielded the highest testing accuracy. This chosen solution is then presented in the following section (Section 5.3.2) and subsequently benchmarked against alternative methods for further comparative analysis. This section is divided into two subsections for both used datasets.

5.3.1.1 Training And Testing Results for D5-JS Dataset

As it has been presented in the experimental design in chapter 3 (Section 4.4.3.2), the JavaScript dataset (D5-JS) was divided into chunks of 2,000 records each, with a training-test split of 1600 to 400 records, respectively. This data chunking approach enabled a rigorous evaluation of the performance and robustness of our Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS) model, ensuring that the algorithm was thoroughly tested across various across various data in incremental manner.

The fitness function results corresponding to various iterations for chunks 0, 1, 2, and 3 are illustrated in Figure 5.2, specifically focusing on two evaluation metrics: F1-score and accuracy. Both of these metrics are reported for two sets of results, namely training and testing data. Each iteration corresponds to a different selected feature

selection solution and thus, different model training weights, leading to variable performance measures across iterations. Across all chunks, a clear trend of improving F1-score and accuracy with increasing number of iterations is observed, demonstrating the efficacy of the Tabu Search algorithm implemented in our 2TS-DFS approach. The testing results show a higher degree of fluctuation compared to the training results, a phenomenon often attributed to data distribution and complexity.

Furthermore, a comparative analysis of the performance at the start and end of each curve indicates an improvement. For instance, the testing F1-score and accuracy for chunk 0 begin at 77.5% and 75.25% respectively, while they reach 82.93% and 81% by the end of the iterations for chunk 0, signifying enhanced model performance. Notably, the peak performance is achieved with an F1-score testing score of 86.1% and accuracy of 84.25% at 119th iteration, underscoring the optimization capabilities of our approach. However, there were some fluctuations in performance as the iterations progressed. There were instances where the model achieved relatively high F1-scores but lower accuracy scores, suggesting that it may have struggled with correctly classifying certain instances. This is due to the large solution space of feature selection and the different possible combinations of selected features. Overall, the model's performance was reasonably consistent, with both F1 and accuracy scores stabilizing around certain values after an initial period of fluctuation.

Similarly, in Chunk 1, we observe a pattern of improvement over iterations for both the Accuracy and F1-score metrics. The initial iteration starts with an Accuracy of 86.69% for training and 83.5% for testing, while the F1-score is 87.5% for training and 83.92% for testing. As the iterations progress, there is a general upward trend in

both metrics, indicating an improvement in the model's performance. The peak values of the testing are achieved at iteration 788, with an accuracy of 85.75% and F1-score of 86.11%. For training, the peak values are reached at iteration 203, with an accuracy of 89.5% and F1-score of 90.14%. Although there are some fluctuations and minor dips in the metrics throughout the iterations, the overall trend is positive, suggesting that the model's performance improves with each iteration.

For chunk 2, both the F1-score and accuracy metrics show an overall pattern of improvement over the iterations. The F1-score for training data starts at 95.32% and gradually increases, reaching a peak value of 96.87% at iteration 116. Similarly, the F1-score for testing data starts at 89.39% and also increases steadily, reaching a peak value of 94.47% at iteration 169. The accuracy for training data starts at 95.3% and shows a gradual improvement, peaking at 96.88% at iteration 116. The accuracy for testing data starts at 89.5% and follows a similar pattern of improvement, reaching its peak value of 94.5% at iteration 157. The range of F1-score values for training data is 89.42% to 96.87%, while for testing data it is 80.26% to 94.47%. The range of accuracy values for training data is 88.75% to 96.88%, and for testing data, it is 79.25% to 94.5%. The improvement in both F1-score and accuracy can be attributed to the training of the current chunk leveraging the knowledge gained from the previous generated solution. As the iterations progress, the model refines its selection decision of the features and learns to make better predictions. The iterative nature of the training process allows the model to adjust its parameters and optimize its performance over time. This results in higher F1-score and accuracy values as the model becomes more proficient at classifying the data correctly.

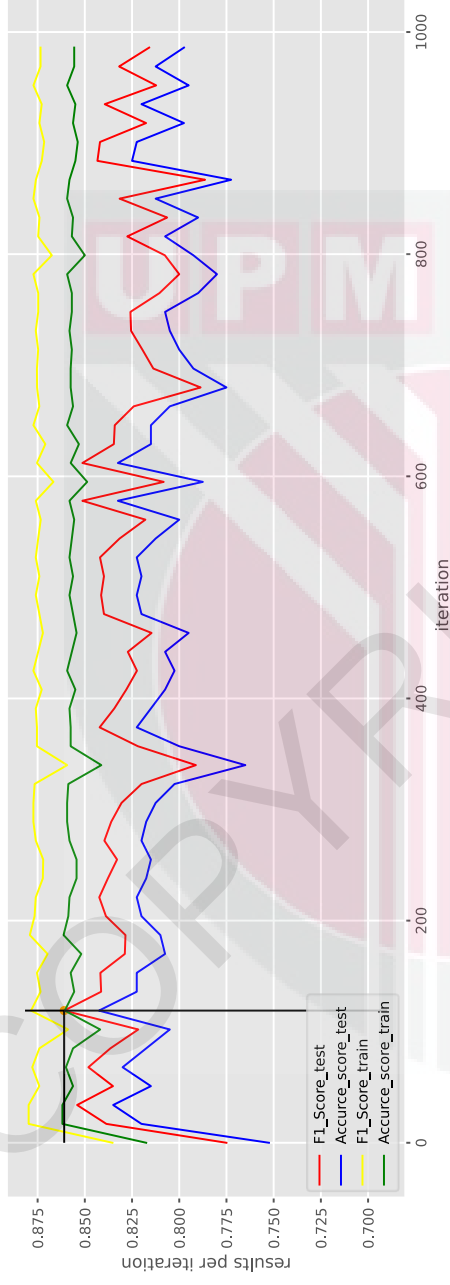
Finally, the last training and testing results for chunk 3 of D5-JS dataset, we also observe a general pattern of improvement in both Accuracy and F1-score metrics for both training and testing data over iterations. The F1 metric for training data starts at 93.7% and gradually increases, reaching its peak at iteration 55 with a value of 95.7%. The F1 metric for testing data starts at 88.34% and also shows a gradual increase, reaching its peak at iteration 666 with a value of 92.6%. The Accuracy metric for training data starts at 94.0% and follows a similar pattern of improvement, reaching its first peak at iteration 55 with a value of 96.0%, then the second peak is reached at iteration 661 with a value of 96.02%. The accuracy metric for testing data starts at 88.5% and reaches its peak at iteration 663 with a value of 93.0%. Overall, we can observe a consistent improvement in both metrics over the iterations. The F1-score ranges from 90.0% to 95.72% for training and from 80.59% to 92.63% for testing. The Accuracy ranges from 90.5% to 96.06% for training and from 82.5% to 93.0% for testing. The metrics show a steady increase, with occasional fluctuations, indicating the model's learning process and ability to generalize well to unseen data.

Overall, even though there might be a variation between the training and testing results, the effectiveness of the testing results is a positive indicator. While overfitting can lead to a drop in test set performance compared to training set performance, the fact that the testing results are still effective suggests that the model has managed to capture meaningful patterns from the training data that can be applied to new, unseen data. In practical applications, achieving high accuracy on the test set, despite potential overfitting, implies that the model is capable of making accurate predictions on real-world instances beyond the training data. It is worth noting that while this phenomenon raises concerns, this work does not delve into resolving this issue as it lies beyond the

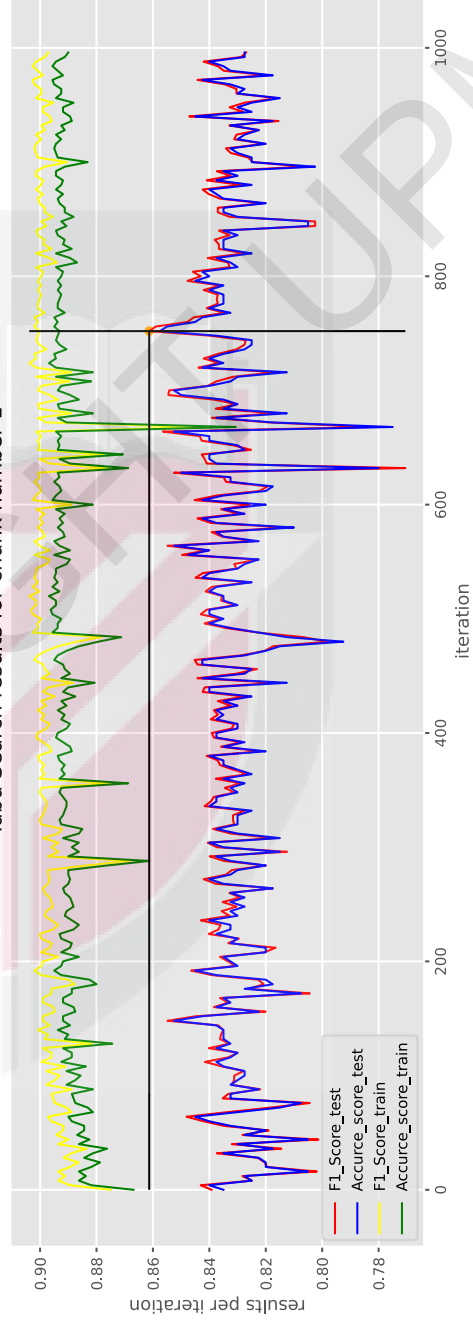
scope of the current research focus, and overall model's predictions are still accurate and effective. Another observation is the increasing trend over the iterative for the different number of chunks reveals the searching capabilities of Tabu search. In other words, tabu search is capable of converging toward an optimal solution for feature selection. On the other side, the fluctuation behavior is an indicator to the complexity of optimization surface as well as the exploration attitude of tabu search.



Tabu Search results for chunk number 0



Tabu Search results for chunk number 1



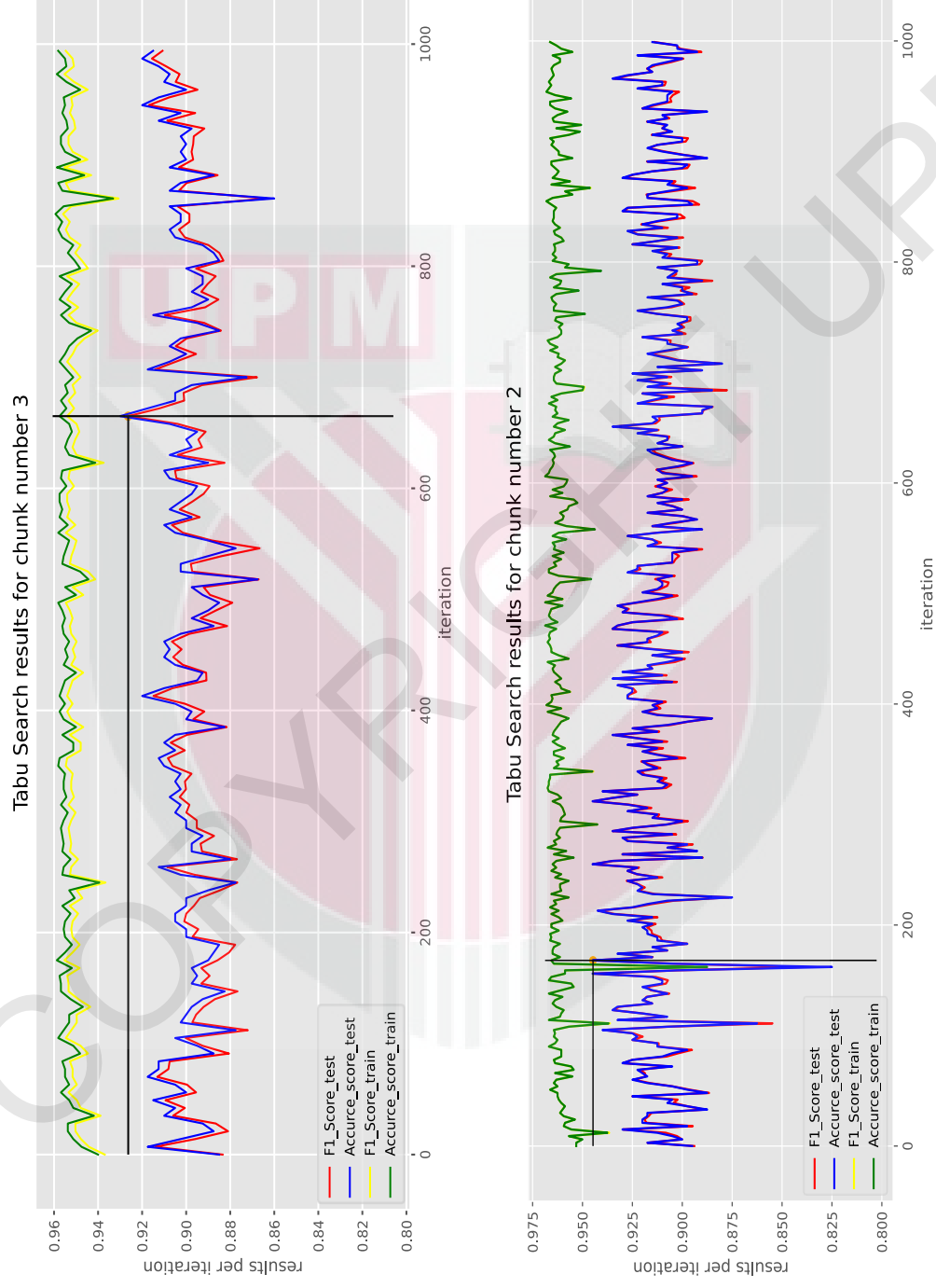


Figure 5.2: Training and testing results of our proposed 2TS-DFS model using D5-JS dataset

5.3.1.2 Training And Testing Results for D6-PL Dataset

The 2TS-DFS model was trained and tested by using another raw dataset which is XSS payloads (D6-PL) dataset, consisting of 10,000 records. This dataset consists of 6503 normal records and 3497 XSS payloads. Similarly, to the previous experiment, the D6-PL dataset was divided into 5 chunks of 2,000 records each, with a training-test split of 1600 to 400 records, respectively. This data chunking approach enabled a rigorous evaluation of the performance and robustness of our Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS) model, ensuring that the algorithm was thoroughly tested across various scenarios and data configurations. The fitness function results corresponding to various iterations for chunks 0, 1, 2, 3 and 4 are illustrated in Figure 5.3, specifically focusing on two evaluation metrics: F1-score and accuracy. Each iteration corresponds to a different selected feature selection solution and thus, different model training weights, leading to variable performance measures across iterations.

For chunk 0, The initial values for F1-score and accuracy were 98.94% and 99.0%, respectively, for training, and 96.72% and 97.0%, respectively, for testing. Throughout the iterations, there is no significant variation in the F1-score and accuracy metrics. They remain relatively stable, with only minor fluctuations. The metrics reach their peak early in the first few iterations and maintain a consistent level thereafter. For training, the F1-score ranges from 98.81% to 99.01%, while accuracy ranges from 98.88% to 99.06%. For testing, the F1-score ranges from 95.16% to 97.0%, while accuracy ranges from 95.5% to 97.25%. Both metrics demonstrated a steady pattern of improvement throughout the iterations. However, since the values remain relatively

constant with only minor fluctuations, it seems that the model selects features that maintain a high level of performance early on throughout the iterations.

The results of chunk 1 over iterations demonstrate a pattern of improvement in both Accuracy and F1-score metrics for training and testing data. Initially, the F1-score for training starts at 98.61% and gradually increases to reach its first peak of 99.08% at iteration 156. Similarly, the F1-score for testing begins at 97.33% and steadily improves until it reaches its first highest value of 98.43% at iteration 481. In terms of Accuracy, the training accuracy starts at 98.69% and steadily rises, peaking at 99.13% at iteration 156. The testing accuracy also exhibits an upward trend, starting at 97.50% and reaching its first peak of 98.13% at iteration 140, then improvement keep continues till it reaches the second peak of 98.5% at iteration 188. The range of values for the F1-score on training data is 98.61% to 99.21%, while for testing data it ranges from 97.33% to 98.43%. The range of Accuracy values for training data is 98.69% to 99.13%, and for testing data, it spans from 97.50% to 98.5%.

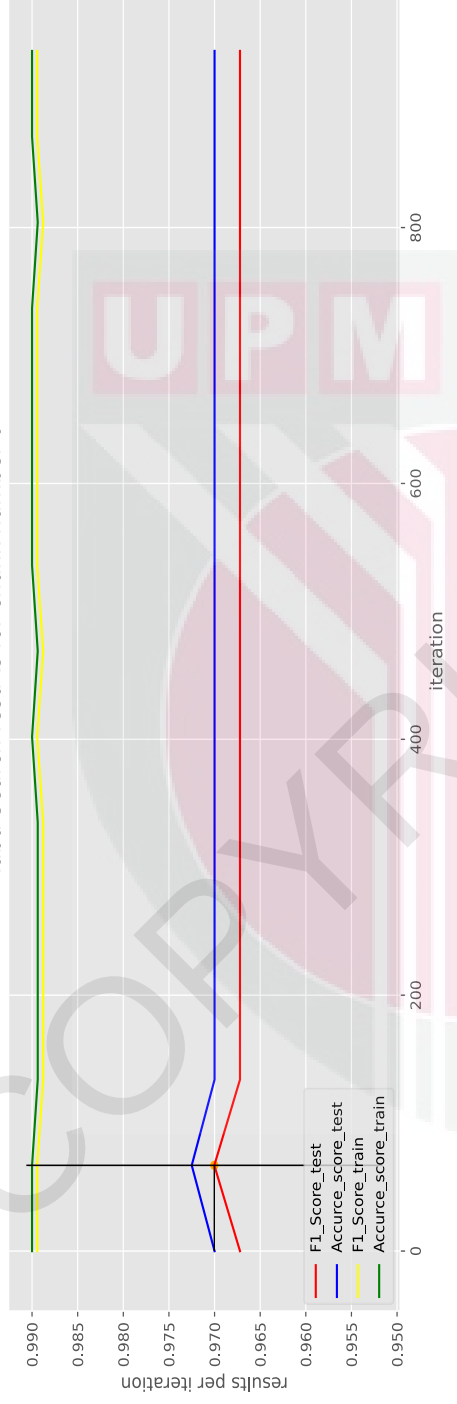
The results of the chunk 2 of iterations show the pattern of improvement in the Accuracy and F1-score metrics for training and testing. Initially, at iteration 1, the Accuracy for training is 98.94% and for testing is 98.25%, while the F1-score for training is 98.96% and for testing is 98.2%. As the iterations progress, the pattern shows a consistent improvement in performance. The training metrics reach their peak at iteration 33, with Accuracy of 99.0% and the F1-score of 99.08%. The testing metrics also exhibit an upward improvement reach their peak at iteration 87, with accuracy of 98.75% and the F1-score of 98.72%. Throughout the iterations, the Accuracy ranges between 98.94% and 99.0% for training, and between 98.25% and

98.75% for testing. The F1-score ranges between 98.9% and 99.08% for training, and between 98.2% and 98.72% for testing. Overall, the pattern of improvement shows a gradual increase in performance measures for both Accuracy and F1-score metrics. The model's performance stabilizes after reaching its peak at iteration 86, indicating that further iterations do not significantly affect the metrics.

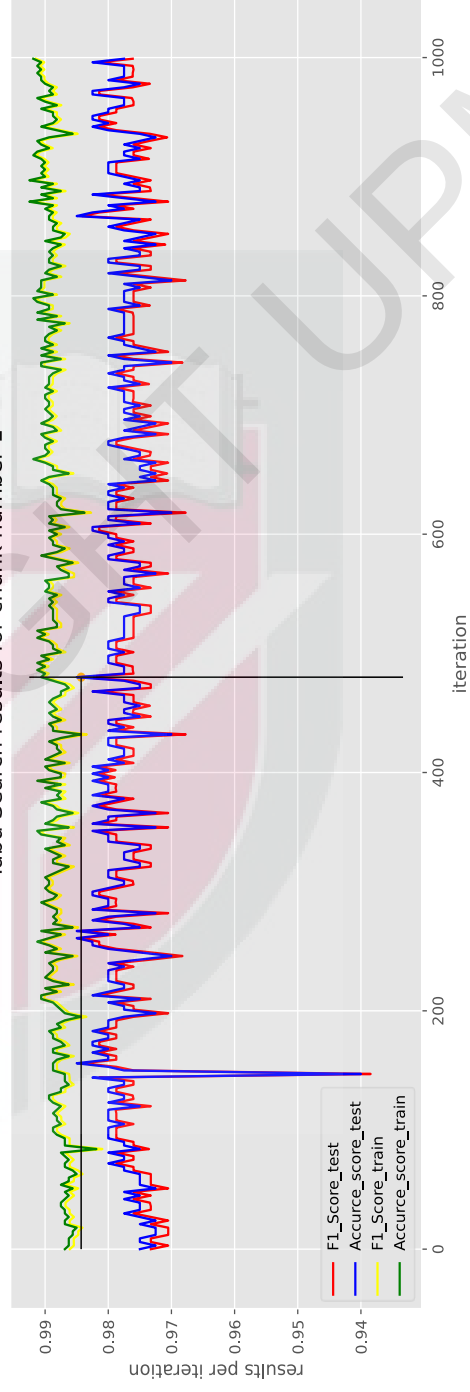
In Chunk 3, we also observed a consistent pattern of improvement over the iterations for both the accuracy and F1-score metrics for both training and testing. The initial iteration starts with an accuracy of 99.06% for training and 98.75% for testing, while the F1-score is 99.07% for training and 98.84% for testing. As the iterations progress, we see fluctuations in the metrics, but an overall upward trend can be observed. The metrics reach their peak at iteration number 83, where the F1-score for training is 99.07% and for testing is 99.3%. The Accuracy at this iteration is 99.06% for training and 99.25% for testing. After reaching the peak, the metrics continue to oscillate around these values without significant improvement or deterioration. The range values for the metrics across iterations indicate that the models trained using different tempo-spatial feature selection solutions have relatively consistent performance. The F1-score ranges from 98.84% to 99.3% for testing, while the Accuracy ranges from 98.75% to 99.25% for testing. Overall, we can observe a general trend of improvement in performance over the iterations, reaching a peak at iteration 83, followed by a plateau in performance. This suggests that the tabu search algorithm effectively explores the feature space and finds a near-optimal solution, leading to improved model performance.

By considering the last chunk 4 of the D6-PL dataset, we also observed a consistent pattern of improvement over iterations for both F1-score and accuracy metrics for training and testing. Initially, in the first few iterations, the F1-score and accuracy values are already quite high for training around 99.0% and F1-score and accuracy for testing around 98.0%. As the iterations progress, we see a slight fluctuation in the metrics, but overall, the values remain relatively stable and close to the initial values. The peak performance for the metrics is reached early on first the iterations, around iteration 101, where F1-score and accuracy reach 99.56% and 99.57% for training, respectively. For testing F1-score and accuracy reach 98.55% and 98.5%, respectively, around iteration 18. These values remain consistent throughout the iterations, with only minor variations.

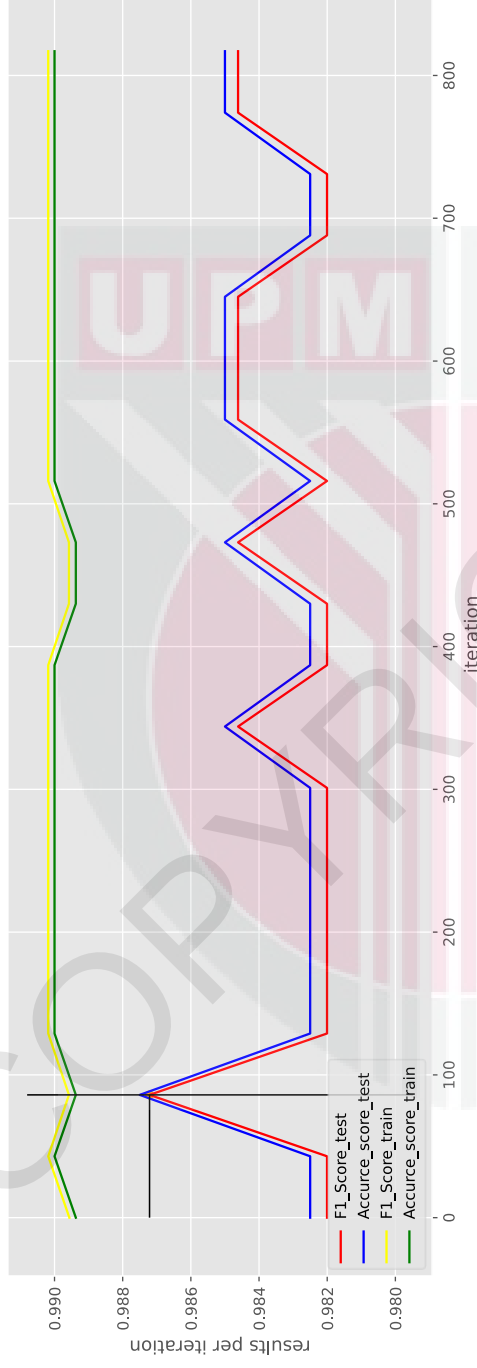
Tabu Search results for chunk number 0



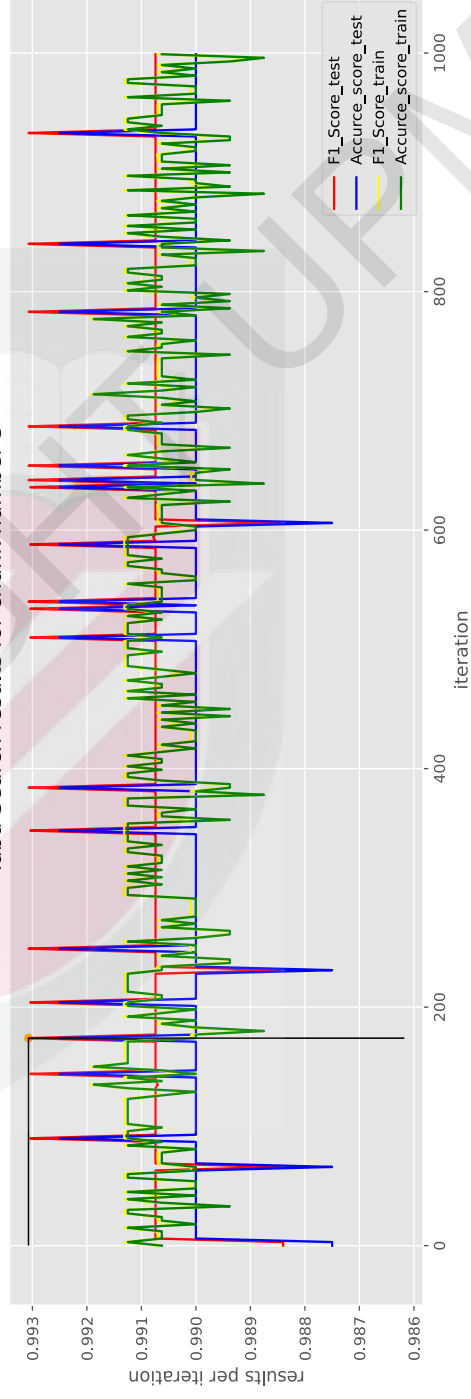
Tabu Search results for chunk number 1



Tabu Search results for chunk number 2



Tabu Search results for chunk number 3



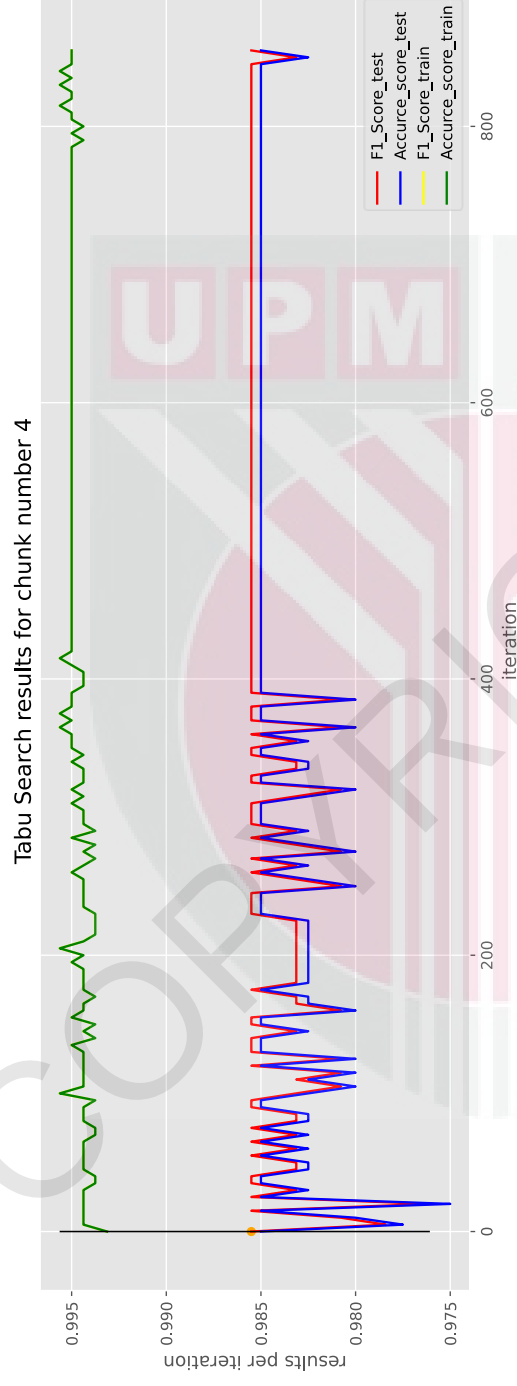


Figure 5.3: Training and Testing results of the proposed 2TS-DFS model using D6-PL dataset

5.3.2 Comparative Analysis

In this section, the performance of the 2TS-DFS model is compared with other existing methods, namely JSContana (Y. Huang et al., 2021), our previously developed RL based dynamic feature selection DQN-MAFS models, and a static word embedding (JSContana-S) method. Initially, the benchmarking was performed against JSContana (Y. Huang et al., 2021) due to its adaptable context analysis method, which integrates word2vec model with a two-layer Bi-LSTM. Subsequently, the comparison was made with the high-performing models of DQN-MAFS, which introduced a temporal dynamic feature selection method using a multi-agent RL approach. Four models were proposed, each model proposed with a unique reward distribution approach, namely accumulative contribution (ACC), alternative contribution (ALC), impurity based (IM), and one action at one time (OA-OT), to distribute rewards among agents based on their actual contributions. In this experiment, only the ACC, ALC, and OA-OT methods that demonstrated superior detection performance were selected. Finally, the proposed 2TS-DFS method was also benchmarked against a static word embedding method, which employed Word2vec model for word embedding representation and TextCNN model for learning and classification tasks. The comparisons are conducted using two raw textual datasets D5-JS and D6-PL that are presented in following subsections.

5.3.2.1 D5-JS Dataset

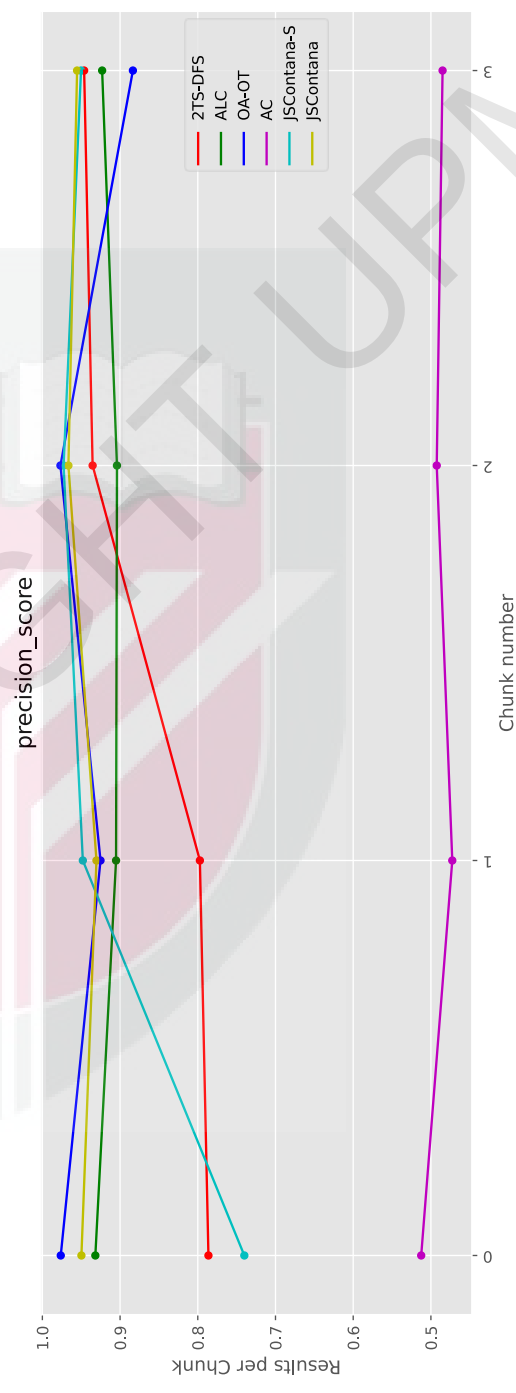
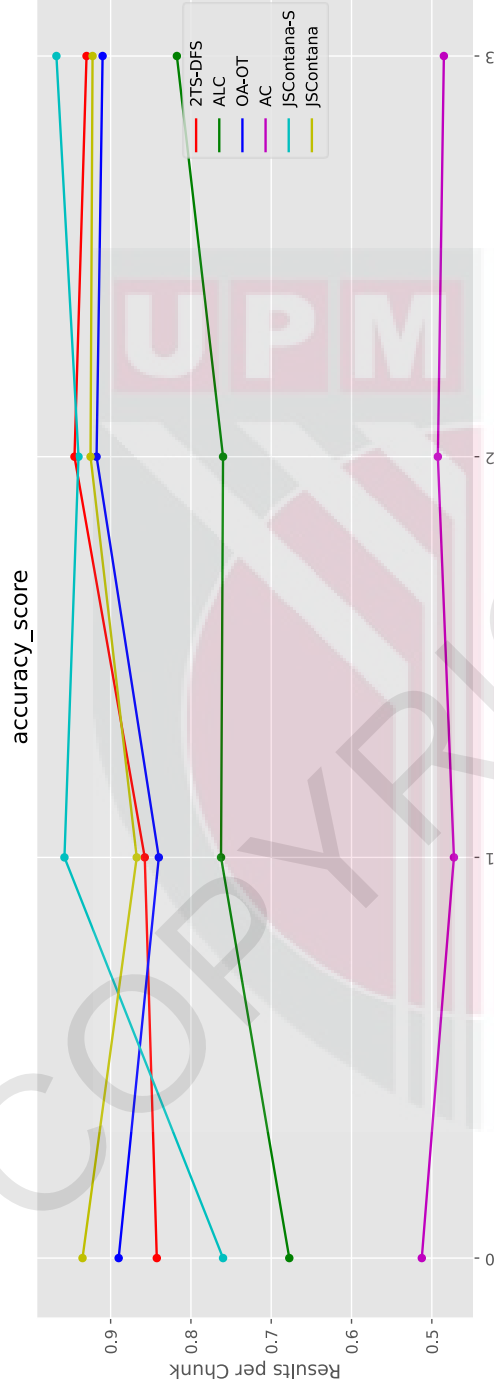
We provide the classification results of 2T-DFS over different chunks of the D5-JS dataset in Figure 5.4. The analysis of the JavaScript dataset D5-JS highlights the

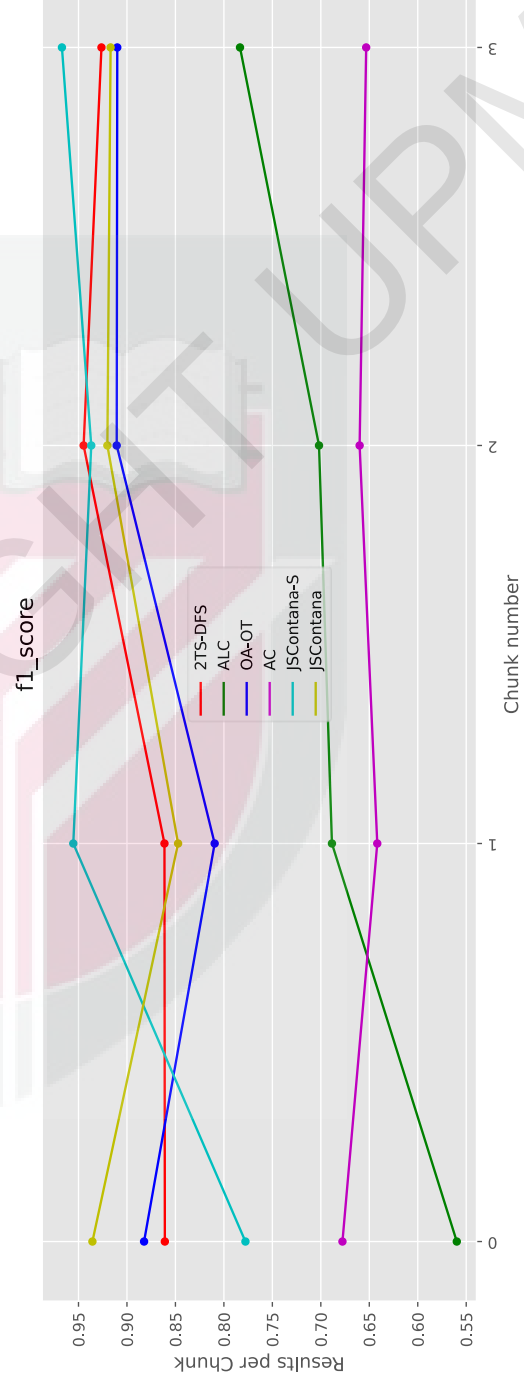
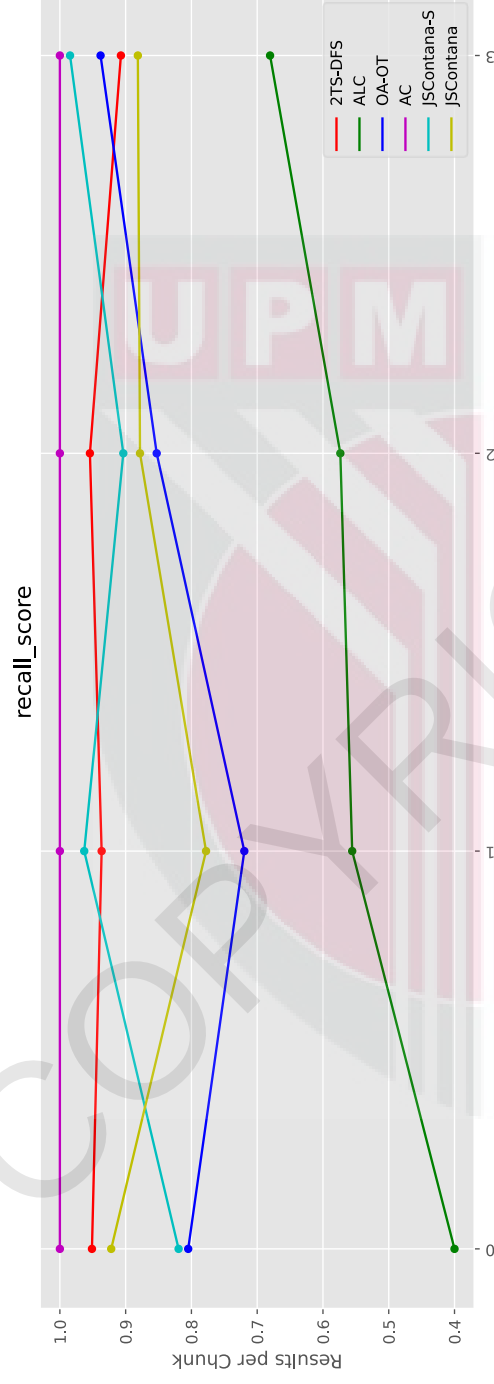
effectiveness of the spatial temporal feature selection conducted by the 2TS-DFS method, which ultimately leads to competitive performance. By dividing the dataset into chunks and evaluating the performance across various scenarios and data configurations, the 2TS-DFS method consistently shows its competitive effectiveness with other methods in terms of accuracy, precision, recall, and F1-score.

In Chunk 1, 2TS-DFS achieves a classification accuracy of 84.25, recall of 95.12%, and F1-score of 86.09%. The False Positive Rate (FPR) is relatively low with 27.1%. ALC shows a lower accuracy of 67.75% with imbalanced precision, recall, and F1-score. OA-OT achieves a slightly higher accuracy of 89% but with a less recall of 80.49%. AC exhibits the lowest accuracy of 51.25% and a FPR of 100%. JSContana-S demonstrates moderate performance with an accuracy of 76% but with a higher FPR of 30.26%. JSContana (Y. Huang et al., 2021) shows a higher accuracy of 93.5% to 2TS-DFS. Moving to Chunk 2, 2TS-DFS improves its performance with an accuracy of 85.75%, and recall of 93.65%, F1-score of 86.13, and its FPR decreased to 21.3%. ALC shows improvement in accuracy of 76.25% but still has imbalanced precision, recall, and F1-score. OA-OT achieves slightly lower accuracy of 84% to 2TS-DFS also with a slightly lower recall and F1-score. AC continue to exhibit poor performance with low accuracy and a high FPR. In comparison with the performance of JSContana, 2TS-DFS shows competitive results but less than the performance of the JSContana-S. Following that in Chunk 3, 2TS-DFS maintains its superiority through high accuracy of 94.5% and demonstrates balanced precision of 93.53%, recall of 95.43%, and F1-score of 94.47. However, the FPR further decreased to 6.40%. ALC achieves a lower accuracy of 76%, precision, recall, and F1-score. OA-OT achieves a higher accuracy of 91.75% with balanced precision and recall. AC continues to exhibit poor

performance, while JSContana-S shows moderate accuracy with an increased FPR of 2.46%. Finally, in Chunk 4, 2TS-DFS continues to perform well with an accuracy of 93%, precision of 94.62%, recall of 90.72%, F1-score of 92.63. The FPR remains decreases to 4.85%. ALC shows improvement in accuracy at 81.75% and achieves balanced precision and recall. OA-OT achieves a similar accuracy with slightly lower precision and a higher FPR of 11.65%. AC exhibits poor performance, while JSContana-S achieves high accuracy with a slightly increased FPR of 4.85%. JSContana performs relatively well with an accuracy of 92.25%, balanced precision, and recall.

In summary, the 2TS-DFS method consistently demonstrates strong performance across the four chunks of the D5-JS dataset. It achieves high accuracy, precision, and recall, indicating its effectiveness in correctly identifying positive instances and it has successfully reduced the misclassification of negative instances as shown in the decreasing pattern of FPR over data chunks. ALC and OA-OT also show improvements in certain metrics but lack consistency across all chunks. AC and JSContana-S exhibit poor performance overall, while JSContana performs relatively well but falls slightly behind 2TS-DFS in some chunks and competitive in another. This highlights the effectiveness of the spatial-temporal feature selection of proposed 2TS-DFS model in identifying and utilizing relevant features that contribute to accurate classification while minimizing false positive errors. In opposite, the other methods not effectively capture the spatial-temporal dynamics present in the dataset, leading to comparatively lower accuracy, precision, recall, and F1-scores, as well as higher false positive rates.





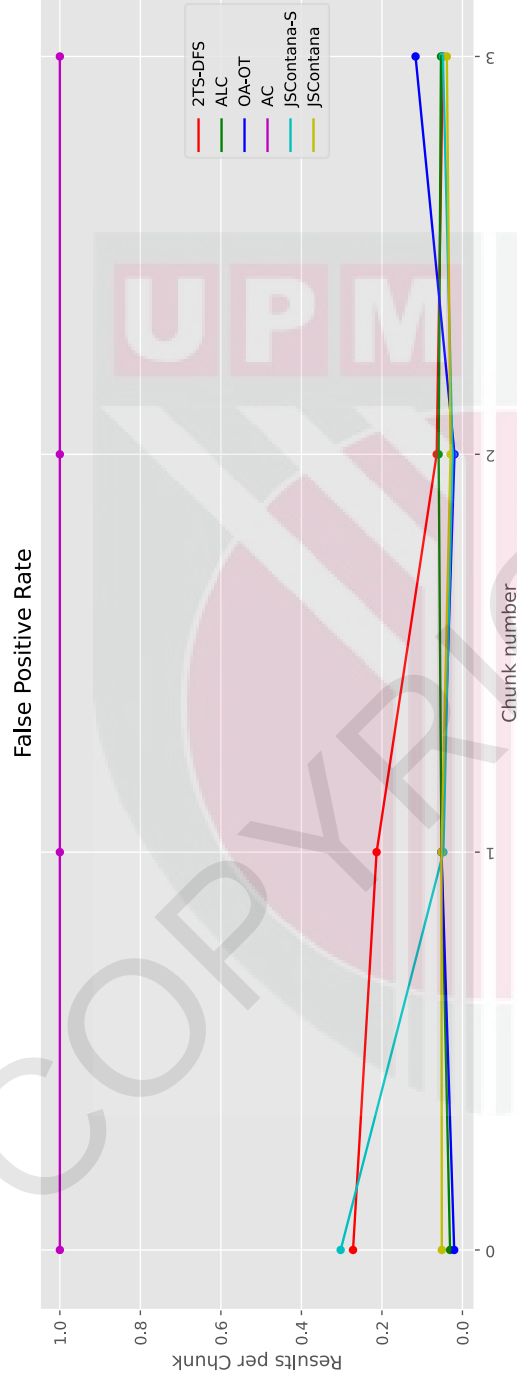


Figure 5.4: Comparison of our proposed 2TS-DFS model with others in terms of classification accuracy, precision, recall, f1-score, and false positive metrics, by using D5-JS dataset

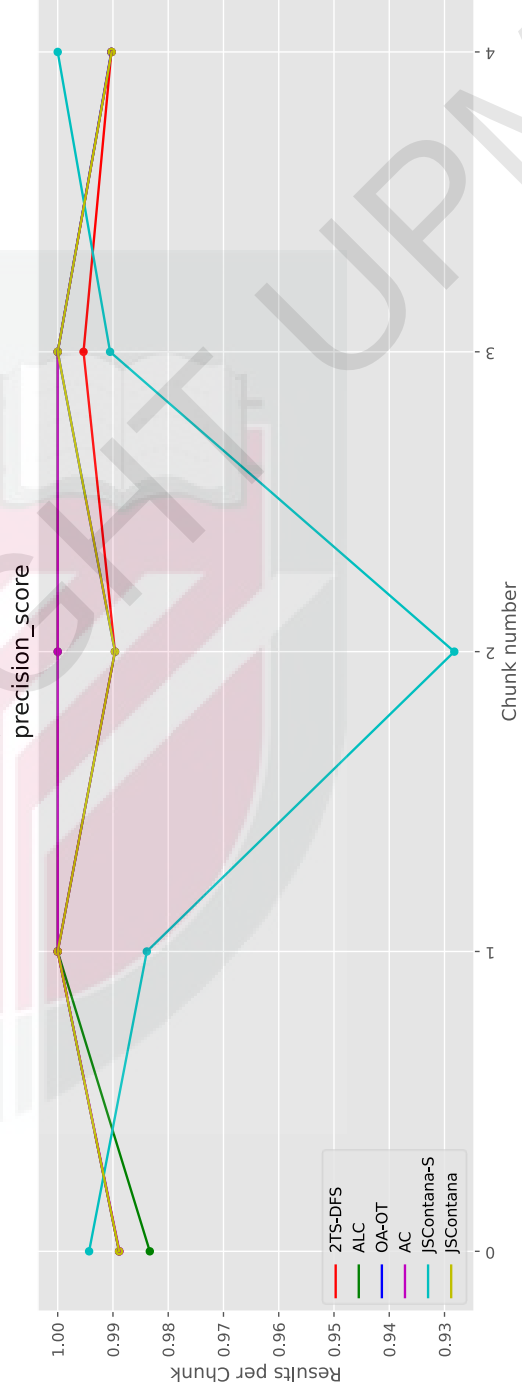
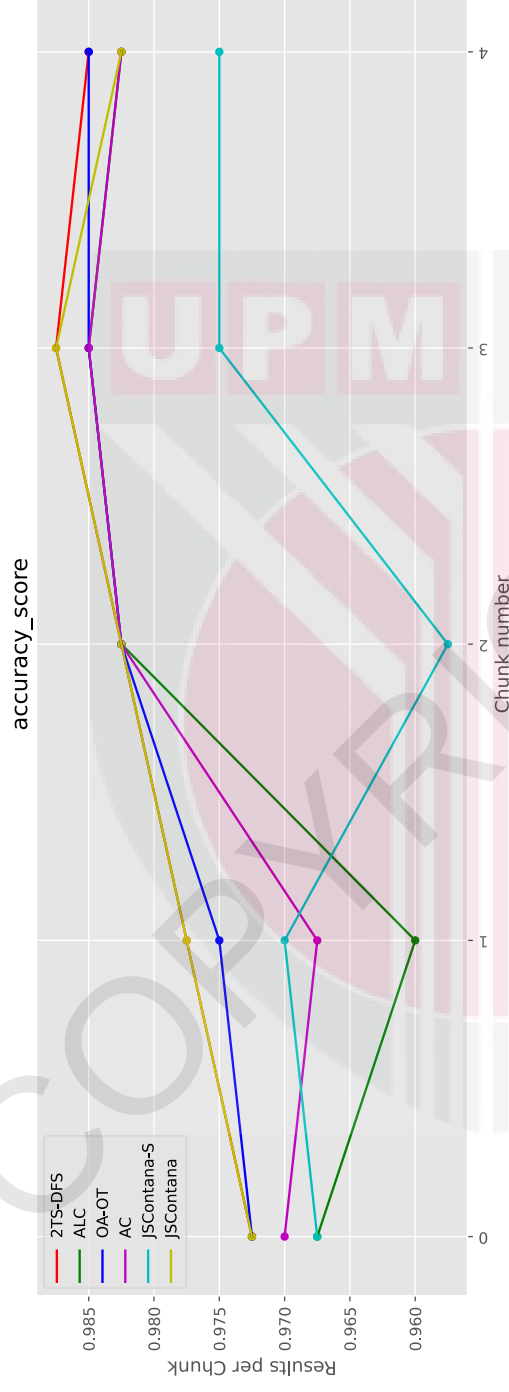
5.3.2.2 D6-PL Dataset

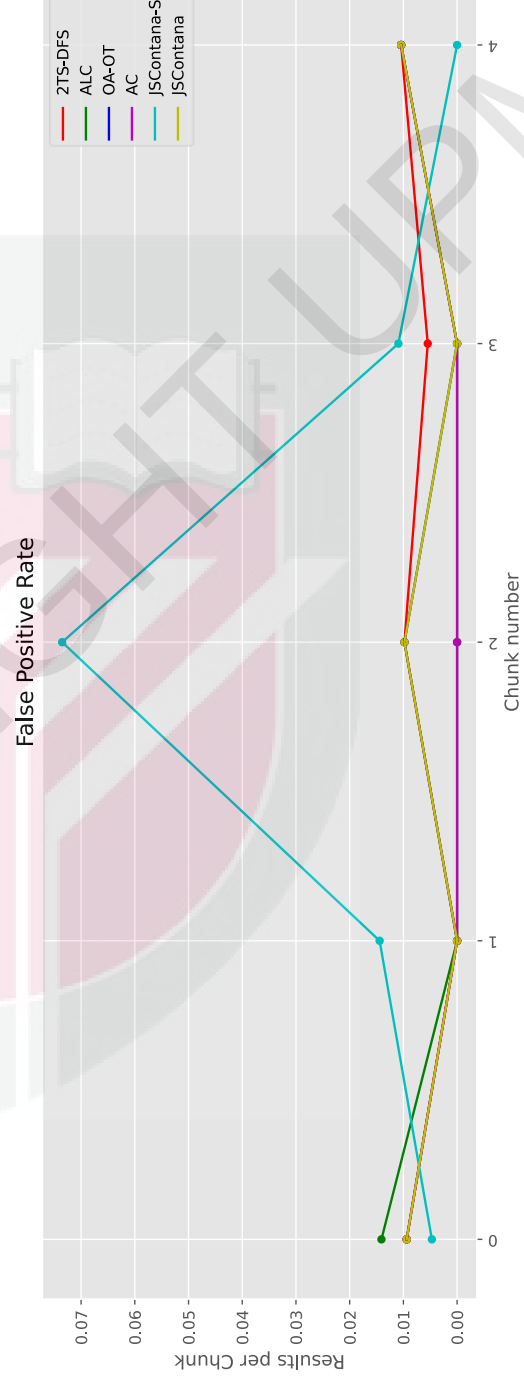
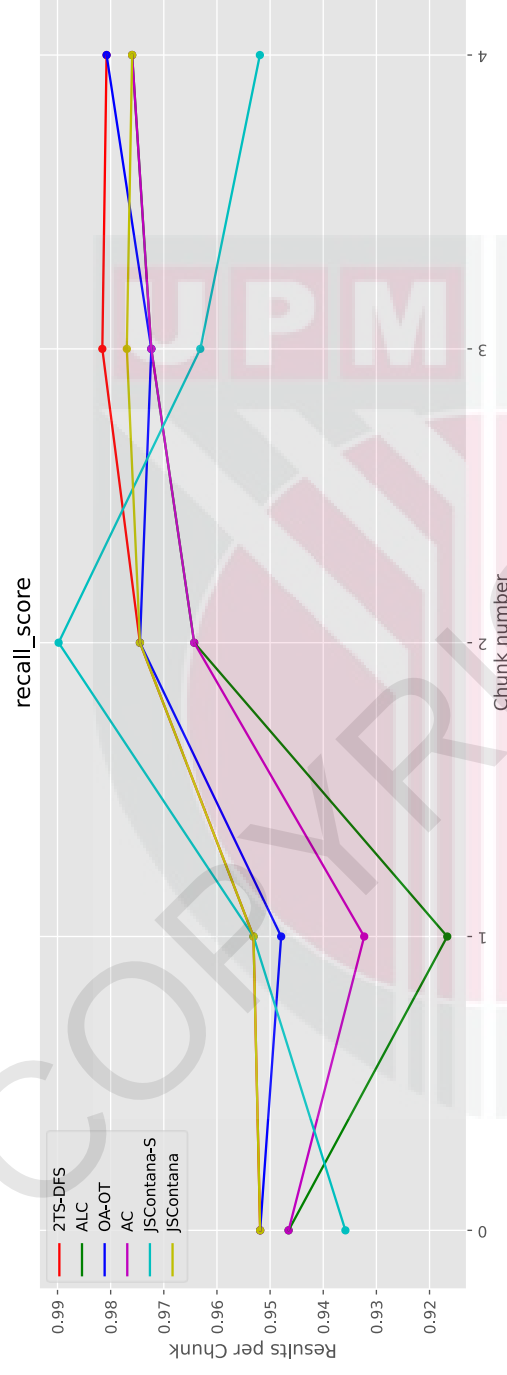
We provide a comparison of the classification results of 2TS-DFS model and other methods over different chunks using D6-PL dataset in Figure 5.5.

In Chunk 1, the 2TS-DFS method demonstrates strong performance with high accuracy (97.25%), precision (98.89%), recall (95.19%), and F1-score (97.00%). It also maintains a low false positive rate (94.00%). ALC and OA-OT achieve roughly similar performance in terms of accuracy and precision, but slightly lower compared to 2TS-DFS, while AC performs well with high precision and recall, but its accuracy and false positive rate are slightly lower than 2TS-DFS. JSContana-S shows a good precision score but falls behind in overall performance, and JSContana shows similar results to 2TS-DFS. Moving to Chunk 2, 2TS-DFS continues to excel with an accuracy of 97.75%, perfect precision, and a high recall of 95.31%. Its F1-score stands at 97.60%, indicating a balanced performance. ALC and OA-OT perform slightly lower in accuracy and recall, while AC achieves comparable results but with a slightly lower precision. JSContana-S, although having a good precision score, but it still falls behind our method. In Chunk 3, 2TS-DFS maintains its superiority with an accuracy of 98.25% and high precision (98.96%), recall (97.45%), and F1-score (98.20%). The false positive rate remains low (0.98%). ALC performs equally well in terms of accuracy and precision but with a slightly lower recall. OA-OT also achieves similar accuracy and precision but with a slightly higher false positive rate. JSContana-S exhibits a lowest accuracy, precision, F1-score, and its high false positive rate indicates a suboptimal performance. In Chunk 4, 2TS-DFS demonstrates its strength once again with an accuracy of 98.75%, precision of 99.53%, recall of 98.16%, and F1-score of

98.84%. The false positive rate remains impressively low (0.55%). ALC, OA-OT, AC, and JSContana achieve comparable results with high precision, recall, and F1-score, but their accuracy is slightly lower compared to 2TS-DFS. JSContana-S performs well in precision but falls behind in accuracy, recall and F1-score. In final chunk (chunk 5), 2TS-DFS continues to demonstrate its effectiveness with a high accuracy of 98.50%, precision of 99.03%, and recall of 98.08%, resulting in an impressive F1-score of 98.55%. In terms of accuracy and F1-score, 2TS-DFS outperforms ALC, AC, JSContana, and JSContana-S, delivering higher values, and performs similarly to OA-OT.

Overall, these results underscore the effectiveness of the 2TS-DFS method in feature selection and classification tasks, with JSContana showing similar results to our method, except in the final chunk where our method surpasses it. Notably, JSContana-S, particularly in the third, fourth, and fifth chunks, falls behind other methods, including ours. Furthermore, our previous method DQN-MAFS also shows outstanding results when compared with JSContana, potentially indicating its suitability for datasets with simplest data distributions. However, the notable difference of 2TS-DFS and JSConatana arises from the dynamic word representation capabilities of these methods. Our 2TS-DFS method showed its capability in capturing the distribution of data effectively over time and dynamic feature selection at the cluster level based on spatial similarities between instances sets our 2TS-DFS apart from our previous DQN-MAFS (AC, ALC, and OA-OT), which selects features only temporally over chunks, allowing 2TS-DFS to excel in capturing evolving patterns and variations of XSS attack with the D6-PL dataset, ultimately leading to improved classification performance.





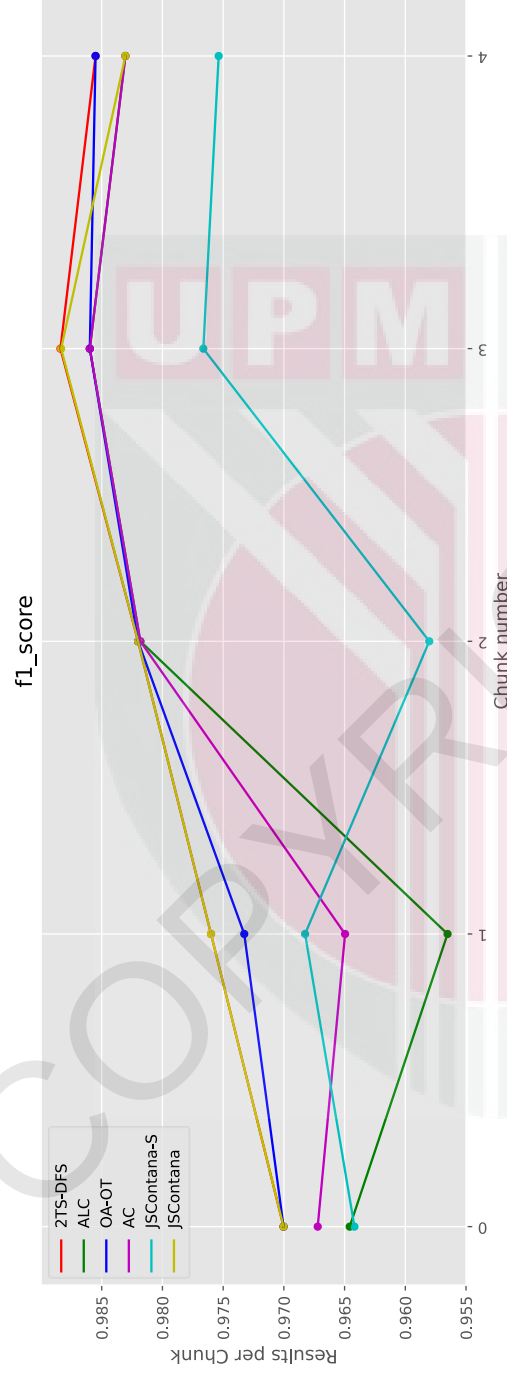


Figure 5.5: Comparison of our proposed 2TS-DFS model with others in terms of classification accuracy, precision, recall, f1-score, and false positive metrics, by using D6-PL dataset

5.3.3 Statistical significance analysis

As an additional assessment step for demonstrating the statistical significance of our proposed 2TS-DFS approaches' results compared to benchmark methods, we conducted some statistical significance tests on depicted results in Figures 5.4 and 5.5. Firstly, t-test results, as presented in Table 5.8, provide a comprehensive evaluation of the statistical significance of the performance differences between our proposed 2TS-DFS method and various benchmark methods (ALC, OA-OT, ACC, JSContana-S, and JSContana) in terms of accuracy and F1-measure on the D5-JS and D6-PL datasets.

Table 5.8: t-test statistical significance testing of the accuracy and F1-measure results in comparison with benchmarks methods

Method	Dataset	T.Test									
		Accuracy					F1-measure				
		ALC	OA-OT	ACC	JSContana-S	JSContana	ALC	OA-OT	ACC	JSContana-S	JSContana
2TS-DFS	D5-JS	0.007	0.818	0.001	0.766	0.513	0.009	0.287	0.002	0.790	0.789
	D6-PL	0.151	0.178	0.108	0.026	0.374	0.159	0.179	0.114	0.020	0.362

For the D5-JS dataset, the t-test results reveal that the 2TS-DFS method significantly outperforms ALC ($p = 0.007$) and ACC ($p = 0.001$) in terms of accuracy, as these p -values are below the significance level of 0.05. This indicates that the differences between 2TS-DFS and these benchmark methods are statistically significant, demonstrating the superior performance of 2TS-DFS. In terms of the F1-measure, the 2TS-DFS method also significantly outperforms ALC ($p = 0.009$) and ACC ($p = 0.002$), reinforcing its effectiveness in identifying relevant features for XSS detection. For the D6-PL dataset, 2TS-DFS shows a statistically significant improvement in accuracy over JSContana-S ($p = 0.026$). Additionally, in terms of the F1measure, 2TS-DFS significantly outperforms JSContana-S ($p = 0.020$).

At the significance level of 0.1, the analysis for the D5-JS dataset shows that 2TS-DFS significantly outperforms ALC ($p = 0.007$) and ACC ($p = 0.001$) in terms of accuracy. The results for the F1-measure further confirm this superiority with significant p -values for ALC ($p = 0.009$) and ACC ($p = 0.002$). For the D6-PL dataset, the accuracy results highlight that 2TS-DFS significantly outperforms JSContana-S ($p = 0.026$). Similarly, the F1-measure results show a significant difference for JSContana-S ($p = 0.020$). The comparisons with ALC ($p = 0.151$), OA-OT ($p = 0.178$), ACC ($p = 0.108$), and JSContana ($p = 0.374$) for accuracy, as well as ALC ($p = 0.159$), OA-OT ($p = 0.179$), ACC ($p = 0.114$), and JSContana ($p = 0.362$) for F1-measure, do not reveal significant differences, indicating that these methods perform similarly to 2TS-DFS. This analysis underscores the competitive nature of 2TS-DFS, demonstrating its significant superiority over specific benchmark methods while maintaining comparable performance with others.

The Shapiro-Wilk test results presented in Table 5.9 provide critical insight into the normality of the distribution of accuracy results for each method across the D5-JS and D6-PL datasets. This test is particularly useful for assessing normality in small sample sizes and helps determine whether parametric or non-parametric statistical methods should be applied for further analysis.

Table 5.9: Shapiro-Wilk Test statistical significance testing of the accuracy results in comparison with benchmarks methods

Datasets	Shapiro-Wilk Test					
	2TS-DFS	ALC	OA-OT	ACC	JSContana-s	JSContana
D5-JS	0.386	0.842	0.448	0.999	0.063	0.130
D6-PL	0.942	0.176	0.190	0.141	0.328	0.959

The Shapiro-Wilk test results, as presented in Table 5.9, indicate that the accuracy results for the methods under consideration generally do not significantly deviate from normality, with p-values exceeding the conventional significance threshold of 0.05. For the D5-JS dataset, all methods, including 2TS-DFS, OA-OT, JSContana-S, ALC, ACC, and JSContana, exhibited p-values ranging from 0.063 to 0.999, which suggests that the data is normally distributed. Similarly, for the D6-PL dataset, the Shapiro-Wilk test results show p-values from 0.141 to 0.959, further confirming the normal distribution of the accuracy results.

Given these findings, it is reasonable to conclude that the results for both the D5-JS and D6-PL datasets is normally distributed, justifying the use of parametric tests such as the t-test for statistical analysis is fairly enough. Consequently, non-parametric tests like the Friedman test were not required, as there was no significant evidence of non-normality in the data distributions.

5.3.4 Memory Consumption

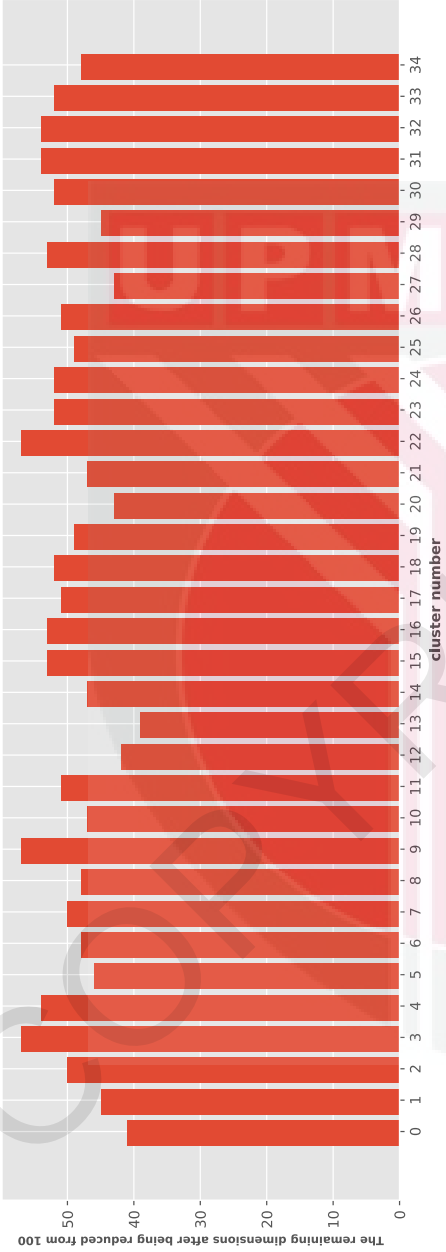
After the clustering process, solution spaces are generated for 35 clusters of each chunk. Therefore, for each chunk, N_c solution spaces are generated, where each solution space corresponds to a cluster within that chunk. These solution spaces capture the relevance of features within their respective clusters, allowing the subsequent searching algorithm to explore and optimize the feature selection process within each cluster independently. By generating solution spaces for each cluster of each chunk, the feature selection algorithm can effectively adapt the selection process to the characteristics and patterns present in different clusters, enhancing the accuracy and efficiency of the overall feature selection process. With this approach, our method aims to achieve temporal and spatial dynamic feature selection to address the feature drift of XSS evolving attacks. The dynamic nature of the feature selection process enables the system to adapt to changing attack patterns over time. Simultaneously, considering the spatial dynamics allows for capturing variations in attack characteristics across different clusters. By integrating these two perspectives, our method provides a robust defence against evolving XSS attacks.

In the following sub-sections, we present the results of the feature selection process over the clusters that were generated for each chunk, demonstrating the memory efficiency of our method in identifying relevant features.

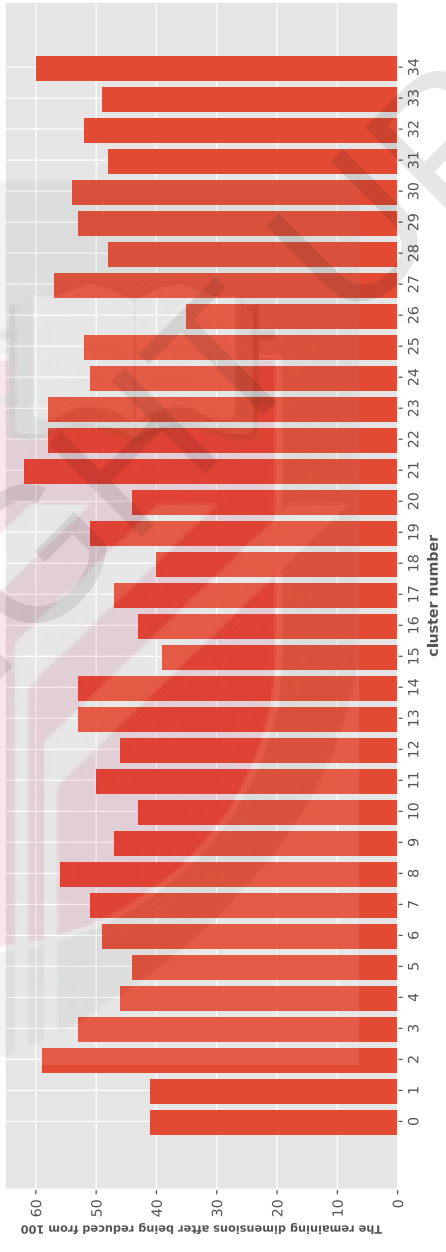
5.3.4.1 Selected Features Over Cluster Using D5-JS Dataset

The feature selection results for the D5-JS dataset are presented in Figure 5.6, showcasing the number of remaining dimensions after feature reduction for each

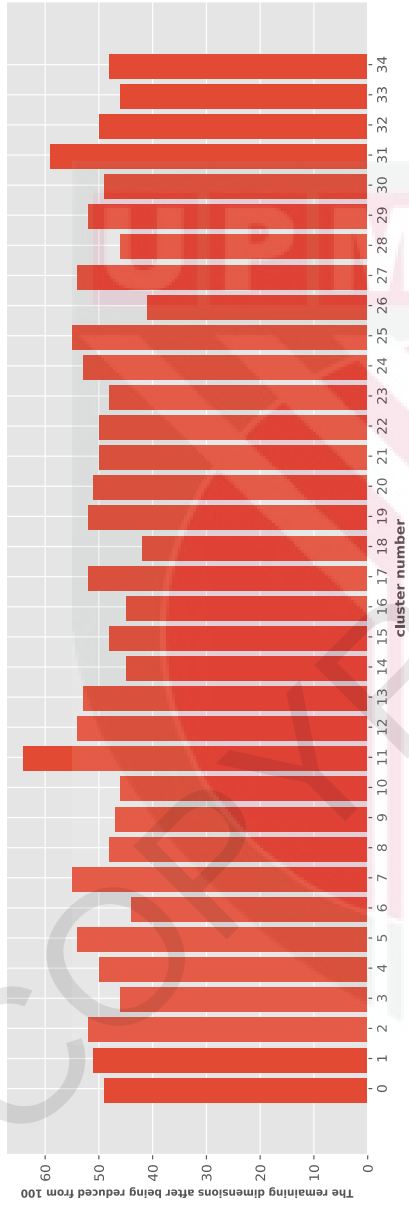
cluster in each chunk. The pattern of feature selection decisions across clusters for each chunk can be observed by analysing the numerical values. In Chunk 0, the feature selection size ranges from 39 to 57, with the majority of clusters having a selection size between 40 and 50. In Chunk 1, the selection size ranges from 35 to 62, with a broader distribution across clusters. Chunk 2 exhibits a selection size range of 41 to 64, with a less variation among clusters. Finally, Chunk 3 shows a selection size range of 41 to 61, with a relatively consistent distribution across clusters. Overall, the feature selection pattern indicates that different clusters within each chunk have varying degrees of feature relevance. Some clusters tend to select a smaller number of features, indicating a higher level of feature concentration, while others exhibit a wider selection, suggesting a more diverse set of relevant features. The range of feature selection sizes highlights the dynamic nature of feature relevance within different clusters, emphasizing the need for a cluster-specific feature selection approach (spatial) to accurately capture the evolving characteristics of XSS attacks.



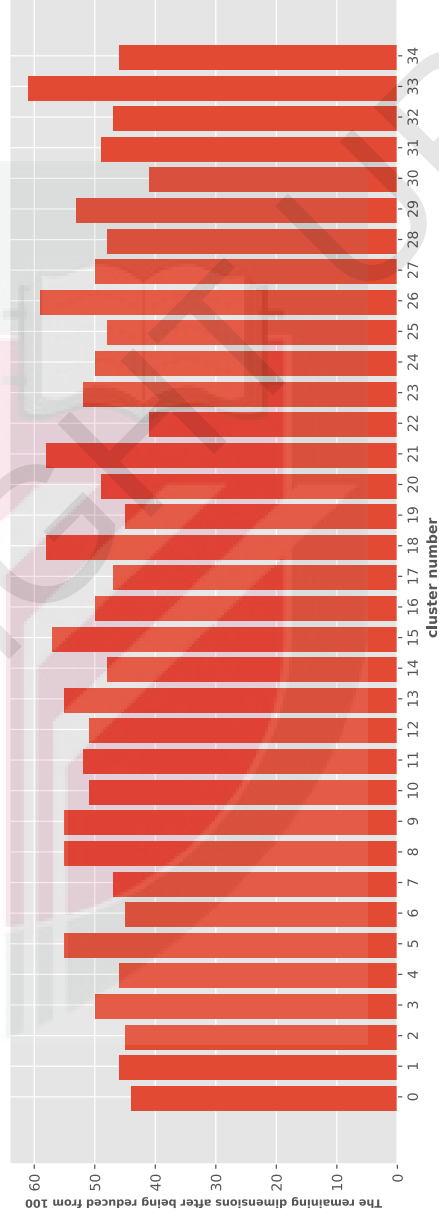
Chunk 0



Chunk 1



Cluster 2



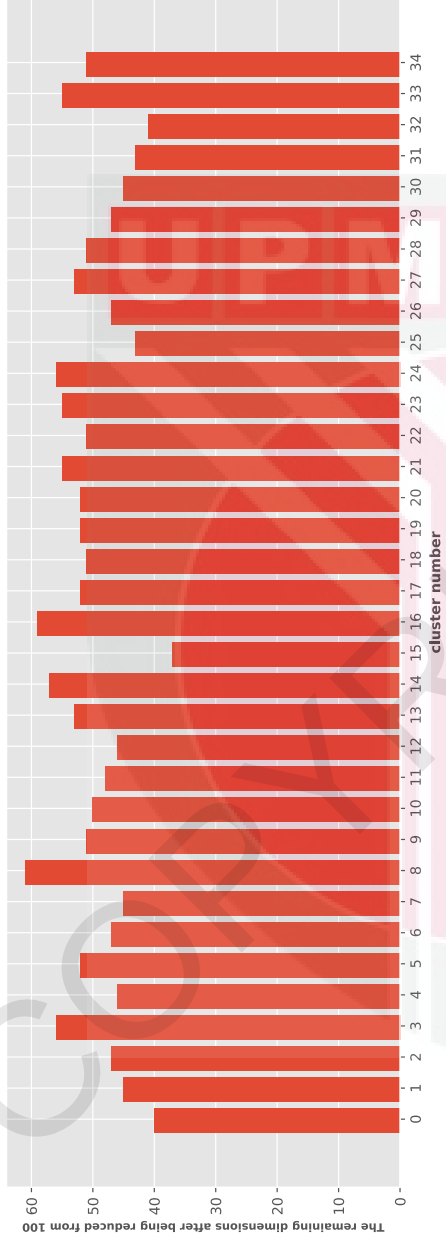
Cluster 3

Figure 5.6: Selected features of 2TS-DFS over clusters for each chunk using D5-JS dataset

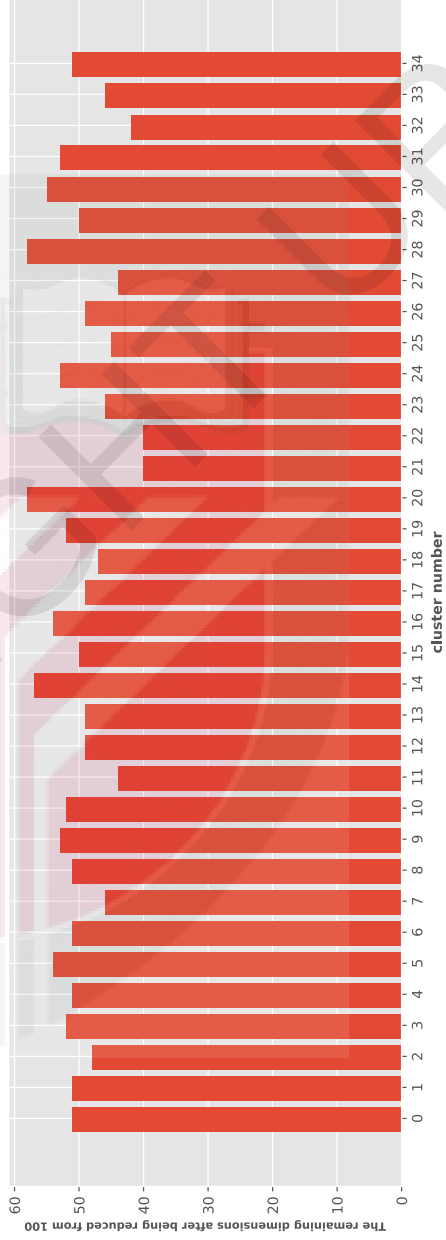
5.3.4.2 Selected Features Over Cluster Using D6-PL Dataset

The feature selection results for the D6-PL dataset are depicted in Figure 5.7, showcasing the number of remaining dimensions after feature selection for each cluster in each chunk. Across the clusters for each chunk in the D6-PL dataset, we can observe distinct patterns in the feature selection decisions. In Chunk 0, the number of remaining dimensions after feature reduction varies from 37 to 61. Most clusters have a selection size between 40 and 51, indicating a consistent emphasis on a subset of features within these clusters. Chunk 1 exhibits a similar trend, with the selection size ranging from 40 to 58. Clusters in this chunk also demonstrate a balanced distribution of selected features. In Chunk 2, we observe a broader range of feature selection sizes, spanning from 40 to 60. This indicates that different clusters in this chunk prioritize a varying number of features, resulting in more diverse feature subsets across the clusters. Chunk 3 displays a relatively narrower range of feature selection sizes, from 42 to 58. However, the specific distribution of selected features differs among clusters, suggesting distinctive feature preferences within the clusters. Lastly, Chunk 4 showcases a relatively consistent pattern, with the selection size ranging from 41 to 62. Most clusters exhibit selection sizes between 45 and 56, indicating a moderate emphasis on an evolving relevant set of features within these clusters spatially and temporally from cluster to cluster and from chunk to chunk.

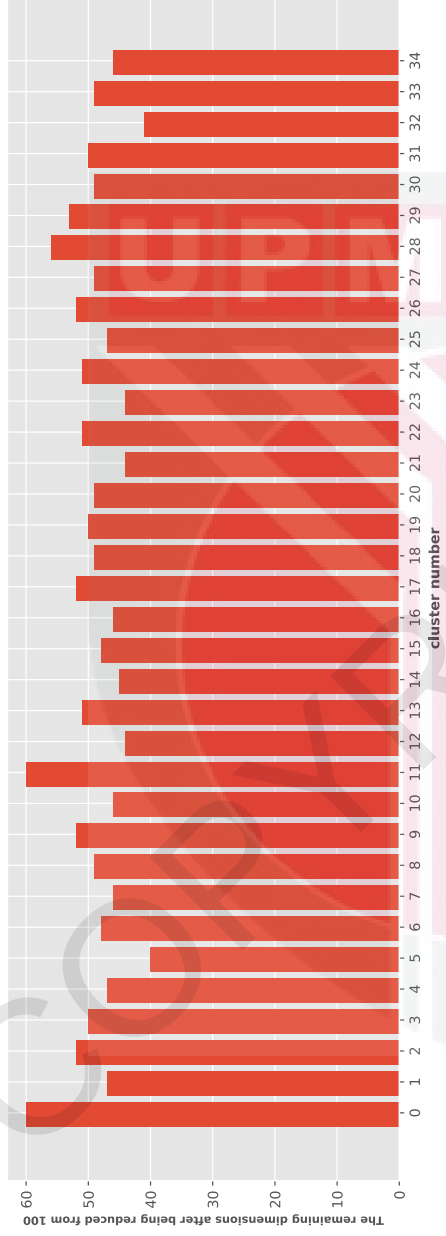
Overall, the feature selection process reveals a mix of cluster-specific and cluster-agnostic feature preferences across the chunks. While some clusters consistently prioritize a particular set of features, others exhibit more varied selections. Again, the range of feature selection sizes emphasizes the need to accommodate diverse feature relevance patterns, allowing for a comprehensive representation of the evolving XSS attacks in both temporal and spatial dimensions.



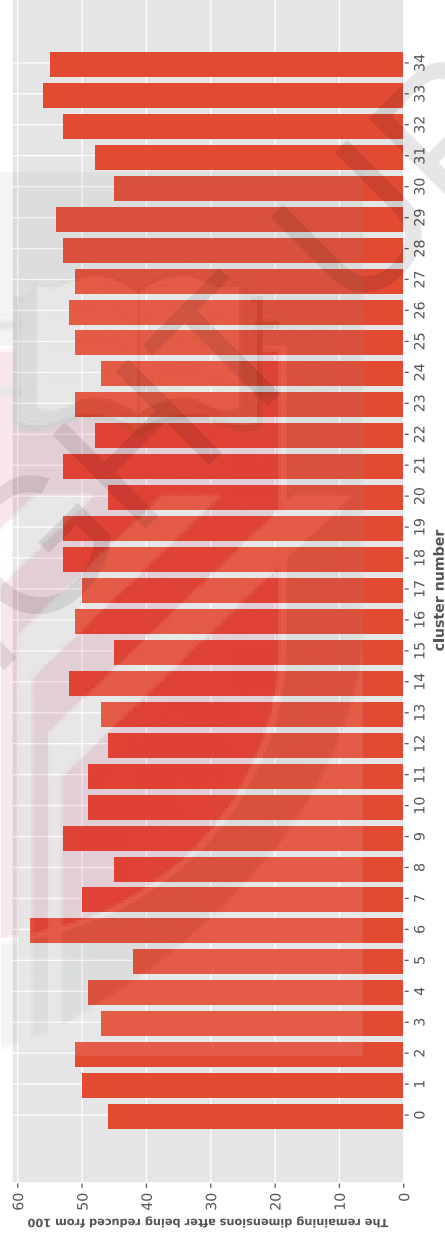
Cluster 0



Cluster 1



Cluster 2



Cluster 3

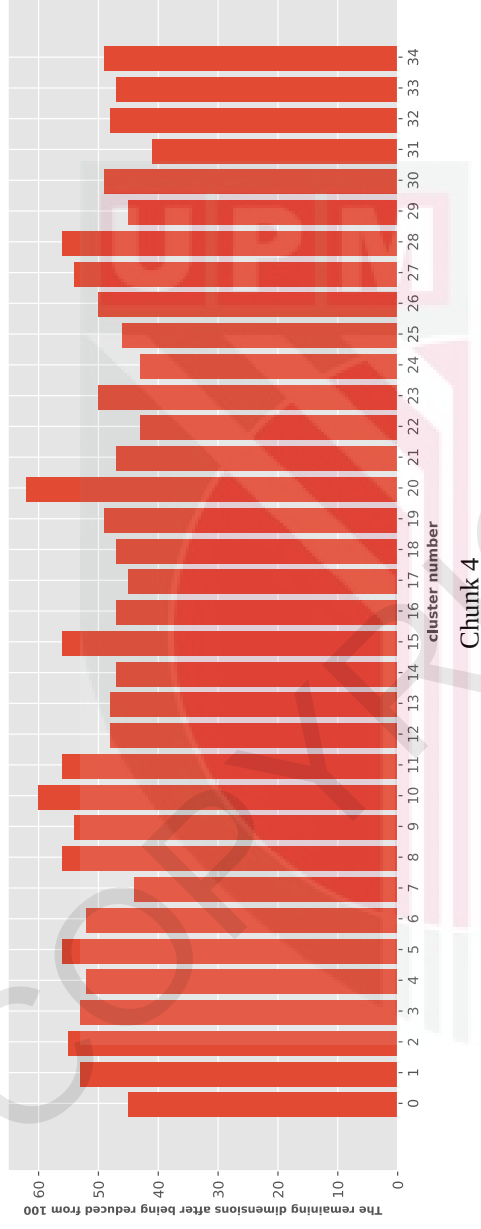


Figure 5.7: Selected features of 2TS-DFS over clusters for each chunk using D5-JS dataset

5.3.4.3 Selected Features of 2TS-DFS Over Chunks in Comparison with Others

In comparing the feature selection capability of the proposed 2TS-DFS method with other methods as shown in Figure 5.8. Firstly, we presented the results of using D5-JS dataset. The results of 2TS-DFS show that the feature selection amounts across the chunks are consistently close to 50, with values of 49.49, 49.51, 49.97, and 50.11 for chunks 0, 1, 2, and 3, respectively. The average feature selection size is 49.78, and the median is 50. The standard deviation is approximately 4.81. The maximum feature selection observed over clusters for the chunks is 64, while the minimum is 35. Comparing these results with other methods, we find that the feature selection amounts for ALC are 44, 55, 42, and 57 across the chunks, while for OA-OT, the values are 59, 47, 39, and 78, and for AC, the values are 52, 52, 56, and 54. Overall, the 2TS-DFS method achieves competitive feature selection results, with feature selection sizes consistently close to 50, indicating a balanced overall selection of relevant features. This is interpreted by the statistical behaviour of Tabu search in 2TS-DFS with respect to 1000 iterations and method of generating neighbouring solution. In comparison, the other methods exhibit more variation in feature selection sizes across the chunks.

Furthermore, we conducted another comparison using the D6-PL dataset, we evaluated the performance of the proposed 2TS-DFS method alongside other methods, namely ALC, OA-OT, and AC. The feature selection results of 2TS-DFS across different chunks showed average sizes ranging from 48.94 to 50.08, with median values of 49 to 51. The standard deviations varied from 3.57 to 5.50, indicating moderate variability. Across clusters over chunks, the maximum feature selection sizes ranged from 58 to 62, while the minimum sizes were between 37 and 42. Comparing the

results of 2TS-DFS with other methods, we found that ALC, OA-OT, and AC exhibited more variation in feature selection sizes across the chunks. ALC had sizes ranging from 40 to 60, OA-OT ranged from 41 to 60, and AC ranged from 43 to 57. In contrast, 2TS-DFS demonstrated a more consistent performance with relatively stable feature selection sizes, suggesting its effectiveness in handling temporal and spatial feature drift. By considering feature selection at the cluster level, 2TS-DFS can detect and adapt to changes in both temporal and spatial features, resulting in more reliable and accurate feature selection outcomes.

In addition to the aforementioned results, it is worth noting that the proposed 2TS-DFS method demonstrates a more consistent feature selection pattern across the chunks. This consistency can be attributed to the fact that our method considers feature selection at the cluster level, taking into account the spatial dynamics of the feature drift and enable this at each chunk, taking into account the temporal dynamic of features selection. On the other hand, the observed variations in feature selection sizes among the other methods can be attributed to their temporal feature selection decisions at the chunk level. This difference in approach highlights the advantage of our 2TS-DFS method in detecting both temporal and spatial feature drift. By considering feature selection at the cluster level, and incremental learning by our method effectively captures the evolving patterns and ensures the selection of relevant features in a more robust and consistent manner.

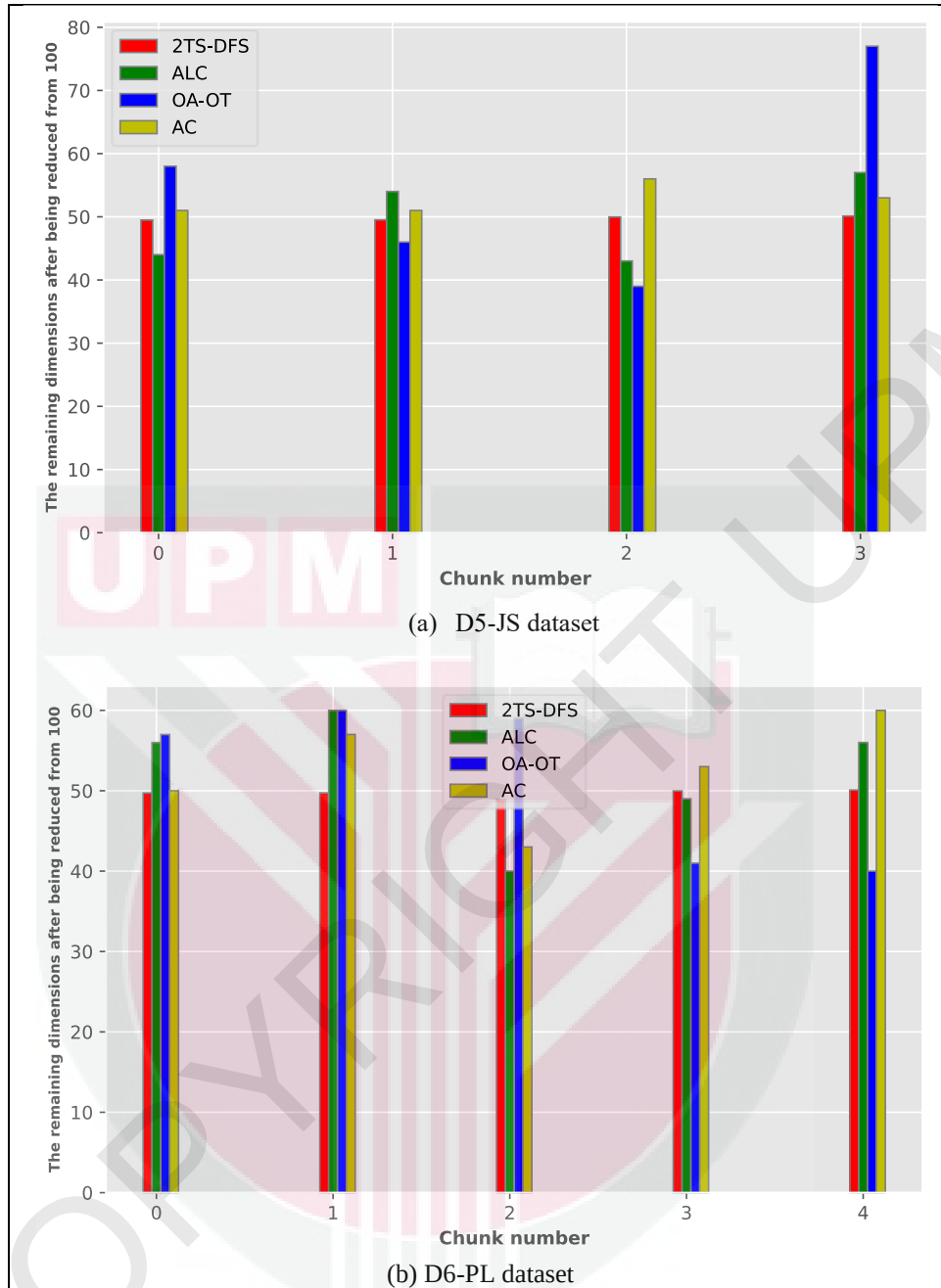


Figure 5.8: Selected features of 2TS-DFS over chunks in comparison with others using D5-JS and D6-PL datasets

5.3.5 Comparisons with other feature selection methods

In our study, we compared our proposed method, Tabu Search based Tempo-spatial Dynamic Feature Selection (2TS-DFS), which incorporates both spatial and temporal feature selection for XSS detection, with other feature selection techniques, including

Lasso, Recursive Feature Elimination (RFE), and Random Forest. The aim was to evaluate the effectiveness of these methods in selecting optimal features for our dataset, which contains high-dimensional feature vectors. Each record comprises 100 features, resulting in a vector with a dimension of 1×2048 , representing part of the hidden state. To address the high dimensionality issue, we transformed these high-dimensional vectors into 1×5 vectors, which encapsulate essential statistics such as minimum, maximum, mean, standard deviation, and median. Subsequently, we applied the feature selection methods to this reduced 5×100 matrix, allowing us to evaluate their performance in conjunction with various machine learning classifiers, including Decision Trees (DT), Gaussian Naive Bayes (NB), Random Forest, K-Nearest Neighbors (K-NN), Logistic Regression, and Support Vector Machines (SVM). We assessed the performance using classification accuracy, F1-score and percentage of selected features as evaluation metrics. This comprehensive comparison helped us determine the effectiveness of 2TS-DFS in the context of dynamic feature selection for XSS detection.

As shown in Table 5.10, in chunk 1, 2TS-DFS outperformed all other methods, including RFE, Lasso and RF, across all classifiers, achieving an accuracy of 84.25% and an F1-score of 86.09%. However, it is worth to note that the RFE and RF showed promising results when combined with the Decision Tree (DT) classifier. In chunk 2, RFE and RF with the DT classifier obtain better results with DT classifier an accuracy of 88.55%, and F1-score of 86.93 for, which makes them competitive with 2TS-DFS that obtained an accuracy of 85.75% and F1-score of 86.13%. Moving to Chunks 3 and 4, our 2TS-DFS method consistently demonstrated outstanding performance, surpassing all other feature selection methods across all classifiers in terms of accuracy

and F1-score. Indeed, the Lasso, RF, and RFE feature selection methods demonstrated an advantage in terms of reducing dimensionality more effectively than 2TS-DFS, as they selected fewer features. However, they failed in selecting the most relevant features to maintain optimal performance and data distribution, and this lack of consistency is a critical issue. On the other hand, 2TS-DFS method consistently outperformed these feature selection approaches in terms of both accuracy and its ability to capture the evolving relevancy of features over time and space. This consistency in performance is a strong indication of the robustness and effectiveness of our approach in dynamic XSS feature selection. It demonstrates the adaptability of 2TS-DFS to spatial and temporal changing patterns and variations in XSS attacks, which is a crucial advantage in real-world scenarios where attack patterns can evolve over time.

Table 5.10: Comparison of 2TS-DFS with other feature selection methods using D5-JS datasets

Feature selection method	Chunk no.	Accuracy					F1-score			Percentage of feature selection	
		KN	DT	LR	NB	SVM	KN	DT	LR	NB	SVM
RF	1	69.50%	81.75%	50.75%	64.00%	65.00%	70.67%	78.84%	7.51%	65.38%	66.02%
Lasso	1	69.25%	62.50%	67.00%	68.75%	63.50%	69.78%	69.01%	68.42%	57.63%	65.07%
RFE	1	69.75%	82.25%	71.25%	68.75%	63.50%	70.27%	79.30%	73.32%	64.59%	65.24%
2TS-DFS	1			84.25%					86.09%		
RF	2	74.75%	88.50%	78.25%	72.25%	71.75%	70.67%	86.93%	79.24%	71.17%	70.50%
Lasso	2	74.50%	71.75%	71.50%	74.50%	71.75%	73.85%	73.78%	70.92%	72.43%	71.25%
RFE	2	74.75%	88.50%	64.75%	75.00%	71.75%	74.04%	86.93%	43.37%	72.83%	71.25%
2TS-DFS	2			85.75%					86.13%		
RF	3	69.75%	92.50%	74.25%	66.25%	65.50%	70.67%	92.02%	76.21%	67.78%	68.92%
Lasso	3	72.25%	65.75%	71.00%	71.25%	66.00%	71.32%	71.04%	72.25%	68.83%	69.64%
RFE	3	71.75%	92.50%	81.75%	72.25%	65.50%	71.10%	92.02%	83.14%	68.01%	69.33%
2TS-DFS	3			94.50%					94.47%		
RF	4	72.75%	90.75%	60.00%	66.50%	65.50%	70.67%	89.92%	30.43%	66.67%	66.34%
Lasso	4	70.50%	63.75%	67.75%	73.25%	66.50%	70.20%	69.47%	68.77%	65.81%	67.94%
RFE	4	71.00%	90.75%	82.25%	74.50%	65.75%	70.71%	89.92%	84.04%	70.00%	67.46%
2TS-DFS	4			93.00%					92.63%		

Additionally, another comparison of 2TS-DFS with other feature selection methods using the D6-PL dataset has been conducted, in this experiment a distinct pattern emerges across different chunks as presented in Table 5.11.

In Chunk 1, RF with various classifiers achieved competitive accuracy and F1-scores, surpassing Lasso and RFE by a significant margin. 2TS-DFS, however, competitive all these methods in terms of accuracy and F1-score. Moving to Chunk 2, RFE and RF maintained their superiority, particularly with DT and NB classifiers. Again, 2TS-DFS achieved high accuracy and F1-scores, remaining competitive. Similarly, in Chunks 3 and 4, 2TS-DFS obtained similar performance with the other methods specifically with DT, LR and NB classifiers, in both accuracy and F1-scores with 89.25% and 98.20%, respectively for chunk 3, and of 98.75% and 98.84, respectively for chunk 4. Finally, in Chunk 5, RF, Lasso, and RFE continued to deliver competitive results to 2TS-DFS, especially with DT and NB classifiers. What stands out is that 2TS-DFS consistently demonstrated competitive performance across all chunks in terms of accuracy and F1-score. Additionally, it is worth noting that RF, Lasso, and RFE selected significantly fewer features than 2TS-DFS, which might explain their varying performance, as they may have missed some relevant features.

Overall, it is important to note that when evaluating RFE, Lasso, and RF with the D5-JS dataset, which has a higher complexity compared to D6-PL, these methods obtained comparatively poor performance. This indicates their limitations in handling more complex data distributions and potentially evolving patterns. In contrast, our method, 2TS-DFS, demonstrated consistency across different data types and over time, showcasing its robustness and adaptability. This adaptability is a significant advantage, especially when dealing with real-world datasets where the complexity and patterns can evolve dynamically.

Table 5.11: Comparison of 2TS-DFS with other feature selection methods using D6-PL datasets

Feature selection method	Chunk no.	Accuracy					F1-score			Percentage of feature selection	
		KN	DT	LR	NB	SVM	KN	DT	LR	NB	SVM
RF	1	97.75%	97.25%	97.25%	97.25%	97.25%	97.56%	97.02%	97.00%	97.00%	97.00%
Lasso	1	76.25%	97.00%	97.00%	97.25%	51.00%	76.07%	96.72%	96.72%	97.00%	40.24%
RFE	1	74.75%	97.25%	97.25%	97.25%	50.75%	74.56%	97.02%	97.00%	97.00%	40.48%
2TS-DFS	1			97.25%					97.00%		
RF	2	98.25%	97.50%	97.50%	98.00%	97.50%	98.14%	97.33%	97.33%	97.87%	97.33%
Lasso	2	78.50%	98.50%	96.75%	98.00%	49.25%	79.02%	98.41%	96.50%	97.87%	40.47%
RFE	2	76.75%	97.50%	97.50%	97.75%	49.25%	76.92%	97.33%	97.33%	97.60%	40.47%
2TS-DFS	2			97.75%					97.60%		
RF	3	98.50%	98.25%	98.25%	98.50%	98.50%	98.46%	98.20%	98.18%	98.45%	98.45%
Lasso	3	78.75%	98.25%	97.75%	98.25%	52.25%	79.62%	98.20%	97.65%	98.18%	50.90%
RFE	3	77.50%	98.25%	98.25%	98.00%	52.25%	78.77%	98.20%	98.18%	97.94%	50.90%
2TS-DFS	3			98.25%					98.20%		
RF	4	98.75%	98.50%	98.25%	98.50%	98.25%	98.83%	98.60%	98.36%	98.60%	98.36%
Lasso	4	76.50%	98.75%	98.00%	98.25%	53.75%	78.24%	98.83%	98.12%	98.36%	52.69%
RFE	4	74.50%	98.50%	98.25%	98.50%	53.50%	76.61%	98.60%	98.36%	98.60%	52.31%
2TS-DFS	4			98.75%					98.84%		
RF	5	98.50%	98.50%	98.00%	98.50%	98.25%	98.55%	98.55%	98.06%	98.55%	98.31%
Lasso	5	80.50%	98.50%	98.00%	98.50%	57.00%	82.19%	98.55%	98.06%	98.55%	54.50%
RFE	5	79.75%	98.50%	98.00%	98.50%	57.00%	81.29%	98.55%	98.06%	98.55%	54.50%
2TS-DFS	5			98.50%					98.55%		

5.3.6 Discussion

In this section, we evaluated the performance of various methods for dynamic feature selection in the context of XSS attack detection. Specifically, we compared our proposed 2TS-DFS method with DQN-MAFS methods including ALC, OA-OT, AC, JSContana-S, and JSContana, using two different raw text-based datasets: D5-JS and D6-PL.

Our analysis focused on key performance indicators, such as accuracy, precision, recall, F1-score, and mean squared error as shown in Table 5.12. Considering the results obtained from the evaluation using the D5-JS dataset, we observed that 2TS-DFS consistently demonstrated superior or competitive performance. It achieved a mean accuracy of 89.4% with a narrow range of values: minimum 84.3% and maximum 94.5%, and a low standard deviation of 0.051. This suggests that our method exhibits robustness and stability in accurately selecting relevant features at cluster level. Comparatively, the alternative methods exhibited varying levels of performance. ALC had a lower mean accuracy of 75.4% with a wider range of values: minimum 67.8% and maximum 81.8%, and a higher standard deviation of 0.058. Similarly, OA-OT showed a mean accuracy of 88.9% with a narrower range: minimum 84.0% and maximum 91.8%, and a lower standard deviation of 0.035. AC had the lowest mean accuracy of 49.1% with a small range of values: minimum 47.3% and maximum 51.3%, and a low standard deviation of 0.017. Furthermore, JScontana-S demonstrates comparative mean accuracy of 90.6% with a high variation range of values between 76.0% and 96.8%, and the highest standard deviation of 0.098. Finally, JSConatana method achieved mean accuracy of 91.3% and high variation range between 86.8%

and 93.5%. These results highlight the consistency and effectiveness of 2TS-DFS in temporal and spatial feature selection over chunks compared to the other methods.

Furthermore, our method exhibited superior performance when evaluated using the D6-PL dataset. It achieved a mean accuracy of 98.1% with a narrow range: minimum 97.25% and maximum 98.75%, and a low standard deviation of 0.006. In contrast, the DQN-MAFS methods demonstrated lower mean accuracies: ALC 97.55%, OA-OT 98.0%, AC 97.75%, JSContana-S 96.9%, and JSContana 98.05%. In terms of F1-score metric, our 2TS-DFS achieved higher mean F1-score of 98.04% with narrow range between 97.0% to 98.84%. While the other methods achieved lower mean F1-scores for ALC, OA-OT, AC, JSContana-S, and JSContana of 97.44%, 97.94%, 97.66%, 96.85%, and 97.99%, respectively. Some of these methods exhibited relatively small ranges of values and low standard deviations, indicating their stable performance. However, 2TS-DFS consistently outperformed them, demonstrating its ability to effectively handle different data types such as JavaScript files (D5-JS) and XSS payloads (D6-PL). The evaluation of other metrics, including precision, recall, F1-score, and mean squared error, further confirmed the superiority of 2TS-DFS over the alternative methods. It consistently achieved higher mean values and lower standard deviations, highlighting its accuracy and reliability in capturing the temporal and spatial dynamics of evolving attacks.

Upon closer examination of the results, it is evident that both the JSContana-S and JSContana models achieved slightly better results than our proposed 2TS-DFS model with some data chunks of D5-JS dataset. However, it is important to note that when comparing the performance on the D6-PL dataset, our 2TS-DFS model outperformed

both JSContana-S and JSContana. Our 2TS-DFS model achieved higher mean accuracy (98.1%), mean precision (99.28%), mean recall (96.84%), mean F1-score (98.04%) compared to JSContana-S and JSContana. This indicates that our model is more robust and consistent in handling feature drift and achieving higher performance across different data types and complexity, surpassing the performance of alternative methods, JSContana-S, and JSContana.

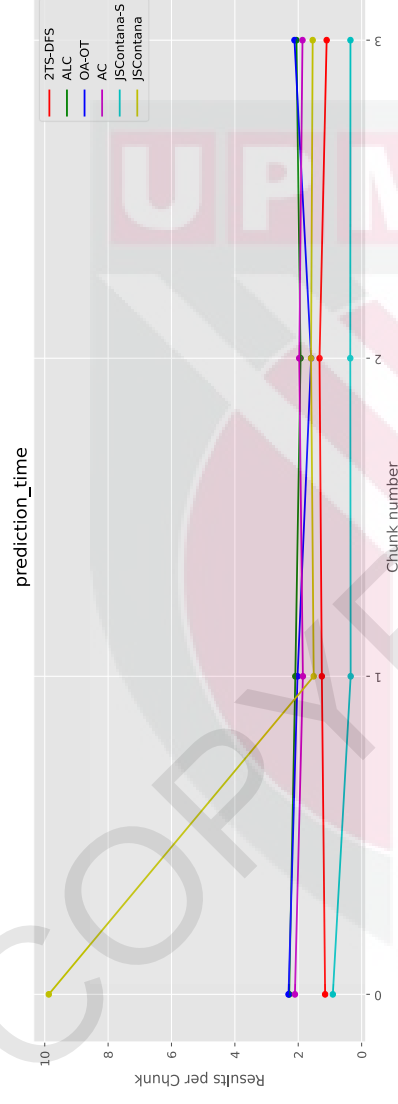


Table 5.12: Overall comparison of 2TS-DFS model by using D5-JS and D6-PL datasets

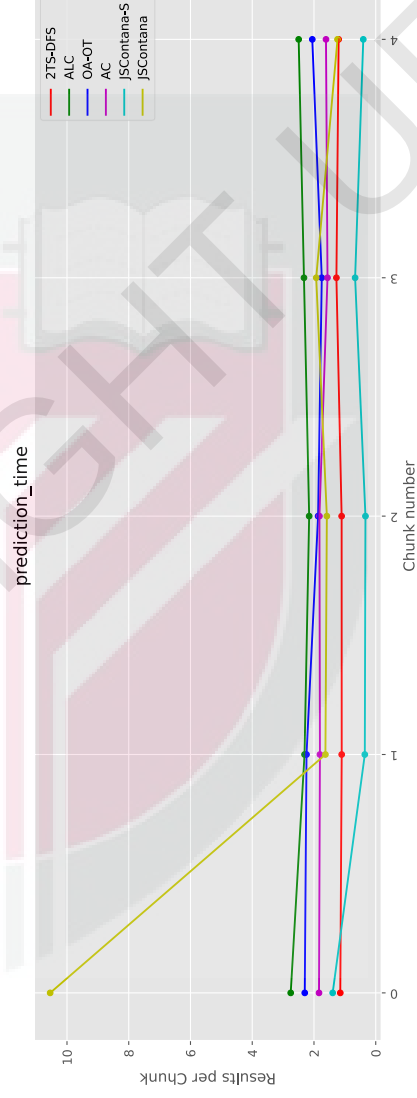
Method name	Metric	D5-JS dataset				D6-PL dataset			
		Mean	Min	Max	Std	Mean	Min	Max	Std
2TS-DFS	Accuracy	89.40%	84.30%	94.50%	0.051	98.10%	97.25%	98.75%	0.006
ALC		75.40%	67.80%	81.80%	0.058	97.55%	96.00%	98.50%	0.0111
OA-OT		88.90%	84.00%	91.80%	0.035	98.00%	97.25%	98.50%	0.0059
AC		49.10%	47.30%	51.30%	0.017	97.75%	96.75%	98.50%	0.0081
JSContana-S	Precision	90.60%	76.00%	96.80%	0.098	96.90%	95.75%	97.50%	0.0072
JSContana		91.30%	86.80%	93.50%	0.03	98.05%	97.25%	98.75%	0.0057
2TS-DFS		86.60%	78.60%	94.60%	0.086	99.28%	98.89%	100%	0.0047
ALC		91.60%	90.40%	93.20%	0.014	99.47%	98.33%	100%	0.0076
OA-OT	Recall	94.00%	88.30%	97.70%	0.045	99.38%	98.89%	100%	0.0057
AC		49.10%	47.30%	51.30%	0.017	99.58%	98.88%	100%	0.0058
JSContana-S		90.30%	74.00%	97.30%	0.109	97.94%	92.82%	100%	0.0292
JSContana		95.00%	93.00%	96.60%	0.015	99.38%	98.89%	100%	0.0057
2TS-DFS	F1-score	93.7%	90.70%	95.40%	0.022	96.84%	95.19%	98.16%	0.0147
ALC		55.20%	40.00%	68.00%	0.116	95.52%	91.67%	97.60%	0.0243
OA-OT		82.90%	72.00%	93.80%	0.091	96.55%	94.79%	98.08%	0.0146
AC		100%	100%	100%	0	95.83%	93.23%	97.60%	0.0184
JSContana-S	Mean Squared Error	91.80%	82.00%	98.50%	0.074	95.88%	93.58%	98.98%	0.0199
JSContana		86.50%	77.80%	92.20%	0.061	96.65%	95.19%	97.70%	0.0128
2TS-DFS		89.80%	86.10%	94.50%	0.044	98.04%	97.00%	98.84%	0.0074
ALC		68.30%	56.00%	78.30%	0.092	97.44%	95.65%	98.60%	0.013
OA-OT		87.80%	81.00%	91.10%	0.048	97.94%	97.00%	98.60%	0.0073
AC		65.80%	64.20%	67.80%	0.015	97.66%	96.50%	98.60%	0.0098
JSContana-S		90.90%	77.80%	96.70%	0.089	96.85%	95.80%	97.66%	0.0078
JSContana		90.50%	84.70%	93.60%	0.039	97.99%	97.00%	98.83%	0.007
2TS-DFS		8.90%	5.10%	12.30%	0.036	1.87%	1.13%	2.72%	0.0062
ALC		15.90%	13.50%	17.40%	0.017	2.13%	1.43%	2.99%	0.0073
OA-OT		9.10%	6.10%	15.20%	0.041	1.98%	1.37%	2.79%	0.0068
AC		25%	25%	25%	0	2%	1.36%	2.78%	0.0069
JSContana-S		7.10%	2.40%	17.80%	0.073	2.87%	2.43%	3.64%	0.005
JSContana		7.40%	5.70%	10.20%	0.02	1.84%	1.15%	2.69%	0.006

Additionally, we also evaluated the prediction time of our method, 2TS-DFS, along with other methods using the D5-JS and D6-PL datasets as illustrated in Figure 5.9. In the case of the D5-JS dataset, we observed that our method exhibited consistently lower prediction times across different chunks compared to ALC, OA-OT, and AC methods. Specifically, the prediction times for 2TS-DFS ranged from 1.103 to 1.328 seconds, whereas the other methods took longer, with ALC ranging from 1.926 to 2.285 seconds, OA-OT ranging from 1.592 to 2.306 seconds, and AC ranging from 1.853 to 2.105 seconds. Notably, JSContana-S demonstrated the lowest prediction times among all methods, with values ranging from 0.345 to 0.908 seconds. Similarly, when analysing the D6-PL dataset, our method, 2TS-DFS, showcased improved prediction times compared to ALC, OA-OT, and AC methods. The prediction times for 2TS-DFS ranged from 1.102 to 1.273 seconds, whereas ALC ranged from 2.156 to 2.753 seconds, OA-OT ranged from 1.747 to 2.298 seconds, and AC ranged from 1.562 to 1.837 seconds. JSContana-S again demonstrated the lowest prediction times, with values ranging from 0.333 to 1.394 seconds.

JSContana-S, demonstrated the lowest prediction time because it does not incorporate any layers for extracting dynamic and adaptable context, while our method employs two layers of BI-LSTM models. Despite the similarity of the method adaptable context analysis that used in our approach and JSContana, we outperform JSContana in terms of prediction time. This can be attributed to our feature reduction techniques, which JSContana does not incorporate. By reducing the dimensionality of the features, we achieve faster and more efficient predictions without compromising the general performance of the model. Overall, the results indicate that 2TS-DFS not only effectively handles feature drift but also provides faster prediction times compared to the evaluated methods, except for JSContana-S.



(a) D5-JS dataset



(b) D6-PL dataset

Figure 5.9: Prediction time of 2TS-DFS model over chunks in comparison with others using D5-JS and D6-PL datasets

5.4 DA-XSS-AD Framework

As it has been presented in the Chapter 3 (Section 3.4), Dynamic and Adaptive XSS Attacks Detection DA-XSS-AD is an integrated framework that includes two sub-models, namely, DQN-MAFS and 2TS-DFS. The framework supports two types of data representation, numerical and word embedding. This framework is distinguished from other existing works in the literature by its high model capacity which enables it to handle a diversity of input data that are classified under the main categories: payload-based XSS and JS-file-based XSS to address the different scenarios in which XSS attacks can occur as illustrated in Chapter 2 (Section 2.2.5). In current section, we present the results of DA-XSS-AD based on different types of XSS attacks within one scenario. The results are compared with the benchmarks of K. Liu et al., (2021), JSContana (Y. Huang et al., 2021) and Tariq et al., (2021). In addition, the results are compared with our first objective results DQN-MAFS which was developed for handling dynamic feature selection of numerical data. As it has been presented, DQN-MAFS considers only temporal aspect of dynamic which makes not optimal for handling textual data representation. However, our developed 2TS-DFS handles dynamic feature selection for textual data as it considers both types of feature drift, namely, spatial and temporal. This makes it more optimal for supporting JS and payload data. Therefore, integrating both DQN-MAFS and 2TS-DFS within DA-XSS-AD will trigger different path of processing according to the nature of data. More specifically, it will trigger DQN-MAFS for numerical data and 2TS-DFS for JS and payload data. We select 10 chunks from the used datasets of all data types. The sampling method is performed by using random uniform distribution. The sequence we received is $Seq = \{D5 - JS - Ch1, D6 - PL - Ch2, D3 - N30 - Ch3, \dots, D4 -$

$N30 - Ch9$ }. Table 4.9 provides the classification accuracy results of the DA-XSS-AD framework, sub-models, and other baseline approaches, categorized based on the data types and representations used. The three categories include JavaScript (JS), payload (PL), and numerical (N) data.

5.4.1 Evaluation of DA-XSS-AD

By analyzing Table 5.13 it becomes evident that the DA-XSS-AD framework excels in performance across all data types. In terms of raw textual JavaScript data, we observe that K. Liu et al., (2021) and Tariq et al., (2021) approaches do not support textual data. In contrast, JSContana (Y. Huang et al., 2021) achieved an accuracy rates of 93.5% for D5-JS-Ch1 and 92.5% for D5-JS-Ch3 indicating its support for JavaScript data. Nevertheless, our proposed DA-XSS-AD achieved higher accuracy values, reaching 89% for D5-JS-Ch1 and 94.5% for D5-JS-Ch3. This demonstrates the superiority of our framework in handling JavaScript data for XSS attack detection. Similarly, when it comes to raw textual payload data, K. Liu et al., (2021) and Tariq et al., (2021) approaches do not support this kind of data type and word embedding representation. However, JSContana (Y. Huang et al., 2021) obtained an accuracy values of 97.8% for D6-PL-Ch2 and 98.3% for D6-PL-Ch5. Again, our DA-XSS-AD framework surpasses these accuracy rates, achieving 97.8% for D6-PL-Ch2 and 98.5% for D6-PL-Ch5. These results highlight the effectiveness of our framework in handling XSS-based payload data and outperforming existing approaches.

Regarding numerical data, JSContana (Y. Huang et al., 2021) approach does not support this kind of the data, while K. Liu et al., (2021) achieved accuracy values of

95%, 95.5%, 95.1%, 97%, 94.4%, and 96.7% for D3-N30-Ch3, D2- N167-Ch4, D1-N66-Ch5, D3-N30-Ch6, D2-N167-Ch8, and D4-N30-Ch9, respectively. Similarly, Tariq et al., (2021) approach obtained an accuracy rates of 80.9%, 88.4%, 66%, 92.1%, 94.6%, and 69.9% for the same data chunks. In contrast, our framework achieved even higher accuracies than the K. Liu et al., (2021) and Tariq et al., (2021), ranging from 98% to 99.8% for D3-N30-Ch3, D2-N167-Ch4, D1-N66-Ch5, D3-N30-Ch6, D2-N167-Ch8, and D4-N30-Ch9. This highlights the capability of our framework in effectively handling numerical data extracted from payloads.

Table 5.13: Classification accuracy of DA-XSS-AD framework and sub-models in comparison with other baseline approaches using different input data types and representations

Chunk No.	K. Liu et al., (2021)	JSContana (Y. Huang et al., 2021)	Tariq et al., (2021)	DQN-MAFS with FARD-DFS approaches				2TS-DFS	DA-XSS-AD
				AC	ALC	IM	OA-OT		
D5-JS-Ch1	-	93.5%	-	51.3%	67.8%	-	84.3%	89%	89%
D6-PL-Ch2	-	97.8%	-	96.8%	96%	-	97.5%	97.8%	97.8%
D3-N30-Ch3	95%	-	80.9%	97.5%	97%	96.5%	98%	-	98%
D5-JS-Ch3	-	92.5%	-	49.3%	76%	-	91.8%	94.5%	94.5%
D2-N167-Ch4	95.5%	-	88.4%	96.4%	98.3%	96.5%	98.4%	-	98.4%
D6-PL-Ch5	-	98.3%	-	98.3%	98.3%	-	98.5%	98.5%	98.5%
D1-N66-Ch5	95.1%	-	66%	99.8%	99.5%	99.7%	99.4%	-	99.8%
D3-N30-Ch6	97%	-	92.1%	98.5%	98.5%	98%	99%	-	99%
D2-N167-Ch8	94.4%	-	94.6%	97.8%	91.4%	98.2%	98.2%	-	98.2%
D4-N30-Ch9	96.7%	-	96.9%	98%	97.5%	97.6%	97.7%	-	98%

From the F1-score perspective, Table 5.14 provides the classification F1-score results of the DA-XSS-AD framework, sub-models, and other baseline approaches, categorized by the chunk number. Regarding raw textual JavaScript, JSContana (Y. Huang et al., 2021) achieved an F1-scores of 93.6% and 92% for D5-JS-Ch1 and D5-JS-Ch3, respectively. However, our DA-XSS-AD framework surpasses these F1-scores, achieving 88.2% for D5-JS-Ch1 and 94.5% for D5-JS-Ch3. For raw textual payload data, JSContana (Y. Huang et al., 2021) achieved an F1-scores of 97% for D6-PL-Ch2 and 98.3% for D6-PL-Ch5. Nevertheless, our DA-XSS-AD outperforms JSContana, achieving an F1-scores of 97.6% for D6-PL-Ch2 and 98.6% for D6-PL-Ch5.

Considering numerical data, K. Liu et al., (2021) achieved an F1-scores of 92.2%, 91.4%, 90.6%, 94.8%, 89.3%, 96.8% for D3-N30-Ch3, D2-N167-Ch4, D1-N66-Ch5, D3-N30-Ch6, D2-N167-Ch8, and D4-N30-Ch9, respectively. Similarly, Tariq et al., (2021) approach obtained an F1-scores of 76.5%, 88.8%, 50.6%, 91.4%, 94.3, and 96.8% for the same data chunks. In contrast, our DA-XSS-AD framework achieves F1-scores ranging from 95% to 98.3% for D3-N30-Ch3, D2-N167-Ch4, D1-N66-Ch5, D3-N30-Ch6, D2-N167-Ch8, and D4-N30-Ch9.

Our DA-XSS-AD framework demonstrates versatility and adaptability by effectively handling different data types, including JavaScript, payload, and numerical data, resulting in high accuracy and F1-scores for XSS attack detection. These findings highlight the superiority of our proposed approach over existing approaches and emphasize its model capacity to handle diverse data types.

Table 5.14: F1-scores of DA-XSS-AD framework and sub-models in comparison with other baseline approaches using different input data types and representations

Chunk No.	K. Liu et al., (2021)	JSContana (Y. Huang et al., 2021)	Tariq et al., (2021)	DQN-MAFS with FARD-DFS approaches				2TS-DFS	DA-XSS-AD
				AC	ALC	IM	OA-OT		
D5-JS-Ch1	-	93.6%	-	67.8%	56%	-	86.1%	88.2%	88.2%
D6-PL-Ch2	-	97%	-	96.7%	96.5%	-	97%	97.6%	97.6%
D3-N30-Ch3	92.2%	-	76.5%	96.1%	95.3%	94.5%	96.8%	-	96.8%
D5-JS-Ch3	-	92%	-	66%	70.2%	-	91.1%	94.5%	94.5%
D2-N167-Ch4	91.4%	-	88.8%	93.7%	97%	93.8%	97.2%	-	97.2%
D6-PL-Ch5	-	98.3%	-	98.3%	98.3%	-	98.6%	98.6%	98.6%
D1-N66-Ch5	90.6%	-	50.6%	95%	87.8%	92.7%	90%	-	95%
D3-N30-Ch6	94.8%	-	91.4%	97.5%	97.5%	96.7%	98.3%	-	98.3%
D2-N167-Ch8	89.3%	-	94.3%	96.1%	84.8%	96.8%	96.8%	-	96.8%
D4-N30-Ch9	96.8%	-	96.8%	98%	97.7%	97.8%	97.8%	-	98%

5.4.2 Feature Reduction

To evaluate the feature selection effectiveness of the DA-XSS-AD framework, we use the feature reduction ratio (FRR) metric. The FRR measures the ratio of the number of selected features to the total number of features in the original data as illustrated in Chapter 4 (Section 4.4.1). By computing the FRR, we can assess how effectively the feature selection process reduces the dimensionality of the data. A higher FRR indicates a greater reduction in the number of features, which suggests that the feature selection method is more effective in capturing the relevant information while eliminating irrelevant or redundant features over time. Table 5.15 presents the feature reduction rate (FRR) of the DA-XSS-AD framework, sub-models, and other approaches for the corresponding chunk numbers.

In comparison with others, the works of Tariq et al., (2021) and Zhou & Wang, (2019), did not implement feature reduction for D3-N30 and D4-N30 datasets, resulting in a 0% reduction rate. Similarly, the work of (Y. Huang et al., 2021), did not perform feature reduction for D5-JS. In another work of Mokbal et al., (2021), the authors achieved a notable feature reduction rate of 55% for the D1-N66 dataset by employing a hybrid feature selection approach. While their approach demonstrated improved memory efficiency, it is worth mentioning that the number of selected features remained fixed for the entire dataset, resulting in a static reduction approach. This fixed feature selection strategy may limit the adaptability of the model to changing patterns and variations in XSS attacks over time. In contrast, the DA-XSS-AD framework exhibits a dynamic feature reduction process, where the number of selected features can vary over time. This dynamic approach allows the framework to adapt to

evolving XSS attack patterns and effectively capture relevant information for accurate detection.

By analyzing the FRR results with accuracy and F1-score for the proposed sub-models, we observe that certain models achieved significant reductions in feature size, indicating enhanced memory efficiency but may not perform as effectively in terms of accuracy and F1-scores compared to other models. For numerical data, K. Liu et al., (2021) reduced feature dimension dynamically ranging from 36.70% to 63.60%. In comparison, the DQN-MAFS with FARD-DFS approaches exhibit feature reduction rates for numerical data ranging from 27% to 65% for same chunks. Although K. Liu et al., (2021) approach achieves significant reductions in feature size, their accuracy and F1-scores are lower than the DQN-MAFS across the evaluated data chunks, as explained in the previous section (4.4.1). Thus, our aim with DQN-MAFS approaches is not solely focused on feature reduction because this may not guarantee optimal performance in detecting XSS attacks. Instead DQN-MAFS continuously adapts to changes in feature relevancy to ensure the selection of relevant feature and enhancing the effectiveness and efficiency of XSS attack detection.

On the other hand, for raw textual data JS and PL, the 2TS-DFS approach achieves relatively lower feature reduction rates, with average ranging from 50.09% to 50.51%, but consistently maintains competitive accuracy and F1-scores. While the other DQN-MAFS methods achieved outstanding feature reduction in feature size, ranging from 40% to 61%, but they obtained lower accuracy and F1-scores. This indicates that the selected features are more effective in capturing the relevant patterns for XSS attack detection, leading to better performance compared to other feature reduction

approaches. Thus, the DA-XSS-AD framework selected the most informative feature number for XSS attack detection that achieves a higher accuracy rate and F1-score. This demonstrates that the DA-XSS-AD framework strikes a balance between feature reduction and detection performance. It dynamically reduces the feature set over time, adapting to changing attack patterns, and maintaining high accuracy and F1-scores. By selecting the optimal FRR rate that preserves the highest accuracy and F1-scores, the framework ensures efficiency in memory usage while maximizing the effectiveness of XSS attack detection.

Table 5.15: Feature reduction rate (FRR) of the DA-XSS-AD framework and sub-models in comparison with other approaches

Chunk No.	K. Liu et al., (2021)	JSContana (Y. Huang et al., 2021)	Tariq et al., (2021) and Zhou & Wang, (2019)	Mokbal et al., (2021)	DQN-MAFS with FARD-DFS approaches			2TS-DFS (Avg.)	DA-XSS-AD
					ALC	AC	IM	OA-OT	
D5-JS-Ch1	-				56%	48%	-	41%	50.51%
D6-PL-Ch2	-				40%	43%	-	40%	50.26%
D3-N30-Ch3	36.7%				63.3%	53.3%	53.3%	37%	37%
D5-JS-Ch3	-	No feature reduction was implemented 0% for D3-N30 and D4-N30 datasets	No feature reduction was implemented 0% for D3-N30 and D4-N30 datasets		58%	44%	-	61%	50.03%
D2-N167-Ch4	59%			The features size of D2-N167 dataset was reduced statically by 60%	47.3%	58%	59%	41%	41%
D6-PL-Ch5	-				44%	40%	-	59%	50.09%
D1-N66-Ch5	63.6%				45.5%	56%	48%	55%	45.5%
D3-N30-Ch6	36.7%				23.3%	60%	50%	30%	30%
D2-N167-Ch8	59.3%				45.5%	46.1%	50%	38.9%	50%
D4-N30-Ch9	43.3%				43.3%	60%	33.3%	56%	43.3%

5.5 Overall Discussion

Based on the presented results, the strengths discussion of the DA-XSS-AD framework and its models are illustrated within this section.

The DQN-MAFS sub-model demonstrates dynamic adaptation and temporal dynamic feature selection. By leveraging Deep Q-Networks (DQN) and Multiple Agent Feature Selection (MAFS), it is capable of adaptively learning and selecting relevant features for XSS attack detection. The dynamic adaptation capability allows the model to adapt and update its knowledge based on changing attack patterns and context, ensuring its effectiveness in the presence of evolving XSS attacks. The temporal dynamic feature selection enables the model to capture the temporal characteristics of XSS attacks, enhancing its ability to detect time-dependent attack patterns and evolving threats. The variation in results observed for the DQN-MAFS method can be justified by the unique reward scheme (FARD-DFS) employed in the proposed framework. The reward scheme plays a crucial role in shaping the learning process of the agents and determining their actions. By calculating the contribution of each agent based on its actions (select or deselect), the reward scheme enables the measurement and distribution of reward values in a fair manner. The use of different reward distribution methods, such as Accumulative Contribution (ACC), Alternative Contribution (ALC), Impurity (IM), and One-Action at One-Time (OA-OT), introduces flexibility and adaptability to the learning process. Each method considers different aspects and factors when assigning rewards to the agents, resulting in diverse learning dynamics and decision-making strategies. Due to these variations in the reward scheme, the agents in the DQN-MAFS approach may exhibit different learning patterns, prioritize

different features, and adapt their strategies differently. Consequently, the observed variation in results can be attributed to the diverse ways in which agents perceive and respond to the rewards, leading to different levels of performance and effectiveness in detecting XSS attacks.

On the other hand, the proposed 2TS-DFS method stands out due to its superiority in several key aspects. Firstly, the 2TS-DFS method incorporates adaptable contextual analysis, which plays a crucial role in understanding the context of XSS attacks. By extracting dynamic word vectors through the combination of Word2Vec and Bi-LSTM, the model captures both forward and backward contexts in a sequence to maintain the raw textual context information of XSS attacks. This solves the polysemy problem of code, where the same syntax unit can have multiple meanings. The ability to adaptively analyse the context enhances the model's understanding of XSS attack patterns and significantly improves the accuracy of detection. This dynamic adaptation feature ensures the model's effectiveness in detecting and mitigating new and emerging XSS attacks. Moreover, the 2TS-DFS method considers both temporal and spatial dynamics of XSS attacks. Temporal dynamics are addressed through the Tabu Search-based feature selection, which enables the model to select relevant features based on the current context and evolving attack techniques. This temporal aspect is crucial as XSS attack patterns change over time, and the model needs to adapt accordingly. Additionally, the clustering and cluster assignment steps in 2TS-DFS introduce spatial dynamics by grouping similar records and adapting feature selection based on the specific characteristics of each cluster. This spatial dynamic approach enhances the model's ability to handle variations in XSS attacks across different clusters (at cluster level). Another strength of the proposed method lies in its memory reduction

techniques. By selecting the best subset of features, the model effectively reduces the memory and computational requirements while maintaining high detection accuracy. This feature is particularly valuable when dealing with large-scale data and limited computing resources, as it improves the efficiency and scalability of the XSS attack detection process. Furthermore, While the DQN-MAFS approaches offer dynamic adaptation, temporal dynamics, and memory reduction, the 2TS-DFS method surpasses them by incorporating adaptable contextual analysis, efficiency in prediction, increased model capacity, and memory optimization. These strengths position the 2TS-DFS method as a superior approach for XSS attack detection compared to the DQN-MAFS approaches. However, it is crucial to point out that DQN-MAFS was developed for dealing with numerical features. Hence, while 2TS-DFS is better than DQN-MAFS at dealing with raw textual data, DQN-MAFS is distinct from 2TS-DFS by its capability of handling numerical features of XSS for classification. Consequently, both 2TS-DFS and DQN-MAFS are complementary for providing unique solution for DA-XSS-AD.

The combination of DQN-MAFS and 2TS-DFS within the DA-XSS-AD framework provides a comprehensive solution for XSS attack detection. By considering both temporal and spatial distribution of features, the framework can capture the sophistication and variations of XSS attacks, leading to improved detection accuracy. Additionally, the framework benefits from the memory reduction capabilities of both models, enabling efficient storage and processing of large-scale XSS attack data. Finally, The DA-XSS-AD demonstrates its capacity and adaptability by being able to handle different types of data commonly encountered in XSS attack detection. XSS attacks can manifest in various forms, including malicious payloads embedded within

HTTP request/response messages, and URL links, and JavaScript (JS) files. Each data type presents unique challenges in terms of feature extraction, context analysis, feature selection, and detection.

5.6 Chapter Summary

In this Chapter, we presented the results and discussions of our study on enhancing web application security against XSS attacks using the DA-XSS-AD framework. The framework revealed outstanding performance across various data types and representations, surpassing existing approaches in terms of classification metrics, and feature reduction rates. We observed that the framework effectively handled JavaScript, payload, and numerical data, highlighting its adaptability compared to approaches focused on specific data types. The sub-models, DQN-MAFS and 2TS-DFS, played crucial roles in capturing temporal and spatial feature drift, making accurate predictions, and reducing feature space for improved memory efficiency. Our findings emphasized the importance of selecting an appropriate feature reduction rate to maintain high detection performance. The implications of our research are significant, as the DA-XSS-AD framework provides an efficient and effective solution for detecting evolving XSS attacks, enhancing web application security. Its adaptability, performance, and memory efficiency make it a valuable tool for developers and security professionals.

CHAPTER 6

CONCLUSION AND FUTURE WORKS

6.1 Conclusion

The ubiquity of modern-day online web applications has ingrained them into the fabric of daily life, offering ease of access and generating substantial volumes of data through user interactions. Unfortunately, this prevalence also makes web applications a prime target for cyber-attacks, particularly cross-site scripting (XSS) attacks. These attacks involve malicious scripts embedded within legitimate webpage code, potentially compromising sensitive information and system integrity. The growing prevalence and evolving nature of XSS attacks, led to the challenges of feature drift, that have exacerbated the limitations of existing methods for XSS attack detection. These methods adopted machine learning and deep learning techniques, have shown limitations in scalability and adaptability, particularly performance degradation when encountering XSS attack evolution and feature drift. This degradation in performance in existing benchmarks, highlights the need for a more dynamic and robust approach to handle the dynamic and evolving nature of XSS attacks but also offers mechanisms for continuous knowledge update, temporal and spatial dynamic feature selection. To fill the gap and contribute to the field of cyber security by addressing these challenges, this research introduced the DA-XSS-AD framework, comprising two distinct operational modes: DQN-MAFS and 2TS-DFS. The DQN-MAFS sub-model leverages Deep Q-Networks (DQN) and Multiple Agent Feature Selection (MAFS) for dynamic adaptation and temporal dynamic feature selection. The 2TS-DFS method

incorporates dynamic adaptation, adaptable contextual analysis, spatial and temporal dynamic feature selection based on agglomerative clustering and tabu search. The combination of these methods provides a comprehensive solution for XSS attack detection.

In this thesis, the DQN-based multi-agent features selection (DQN-MAFS) have been proposed for addressing dynamic feature selection for numerical representation in context of evolving XSS attacks. DQN-MAFS model leverages multi-agent learning, where each agent focuses on one feature with two actions, namely, selection or de-selection. By selecting most relevant feature subsets, DQN-MAFS enhances prediction accuracy and efficiency. To tackle the challenges of simultaneous agent actions and non-stationary environment, a sub model FARD-DFS (Fair Agent Reward Distribution based Dynamic Feature Selection) was introduced, comprising method like ACC, ALC, IM, and OA-OT. The experiments of the DQN-MAFS model were performed using four numerical XSS datasets, namely, D1-N66, D2-N167, D3-N30, and D4-N30. The two benchmarks used for comparison are Tariq et al., (2021) and K. Liu et al., (2021). For evaluation, each dataset was partitioned into chunks, similar to how data streams feed incremental learning models. Results Showcased FARD-DFS's superiority over the benchmarks, particularly in scenarios with varying feature sizes. Another observation is that FARD-DFS OA-OT is more superior when the number of features is high around 167 features, achieving a mean accuracy and F1-measure of 97.66% and 95.81%, respectively with D2-N167 dataset. Meanwhile, FARD-DFS-ACC, outperformed other methods and benchmarks in low-feature dataset around 30 features with D3-N30, obtained a mean accuracy and F1-measure of 98.28% and 97.38%, respectively. FARD-DFS-IM has demonstrated strong

performance across medium-feature size and low feature size dataset, achieved mean classification accuracy of 99.59% and mean F1-measure of 88.62% with D1-N66, also achieved highest a mean accuracy of 98.37% and mean F1-measure of 98.42%, with D4-N30. In addition, we found that both K. Liu et al., (2021) and Tariq et al., (2021) have a degradation in the performance when the number of features increases, which means less scalability. This can be interpreted by the powerfulness of the reward calculation and distribution in FARD-DFS. Furthermore, the dynamicity handling of FARD-DFS is consistent and promising when it is used against the evolving nature of XSS attacks in an incremental or stream learning manner. Overall, our proposed approach, DQN-MAFS with FARD-DFS, emerges as a dynamic and effective solution for XSS attack detection in the face of feature drift and evolving attack patterns.

In addition, to response the evolving nature of XSS attacks with raw textual data, this thesis introduced the 2TS-DFS (Tabu Search based Tempo-Spatial Dynamic Feature Selection. This model addresses the need for spatial temporal feature selection through hidden stat representation derived from textual data using Word2Vec model and two layers of Bi-LSTM models. select features spatially and temporally using hidden state representations to detect the evolving nature of XSS attacks. The utilization of dynamic word embedding representation enhances the model's ability to capture dynamic representation for the same feature and to handle evolving attack pattern effectively. The evaluation stage was performed on two raw textual datasets, namely payload data and JS data, we thoroughly examined the 2TS-DFS model's performance. In the case of JS, we use decoding and AST generation. In the case of payloads, we use three steps of pre-processing, namely, decoding, generalization, and segmentation. During training and testing phases, we ensured that our model maintains memory

efficiency and effectiveness while adapting to changing attack patterns to show the efficiency of the temporal-spatial feature selection in reducing feature dimensions. Additionally, the performance comparison were drawn against other existing methods, namely JSContana (Y. Huang et al., 2021), our previously developed DQN-MAFS (ACC, ALC, and OA-OT) models, and a static word embedding (JSContana-S) method. When assessed against other works using the D5-JS dataset, the proposed 2TS-DFS model demonstrated competitive results, effectively managing feature drift. Notably, it outperformed both JSContana-S and JSContana when evaluated using the D6-PL dataset. The model's higher mean accuracy (98.1%), mean precision (99.3%), mean recall (96.8%), and mean F1-score (98.0%) underscored its adaptability and superiority across varying data types. Furthermore, prediction times were rigorously evaluated. The 2TS-DFS consistently exhibited shorter prediction times compared to ALC, OA-OT, and AC methods when considering the D5-JS dataset. Similarly, when analyzing the D6-PL dataset, our 2TS-DFS model showcased improved prediction times compared to the aforementioned methods. JSContana-S achieved the lowest prediction times due to its simpler structure, but our 2TS-DFS model's efficiency and feature reduction techniques allowed it to surpass other methods' prediction times while maintaining robust performance. Overall, the 2TS-DFS model presents a comprehensive solution for handling evolving XSS attacks through spatial-temporal feature selection and hidden state representations. Its impressive results, both in terms of performance and prediction time, affirm its efficacy in addressing dynamic security challenges and enhancing the field of cybersecurity research.

6.2 Future Works

In the light of the performance results of the proposed framework, where it exhibited its efficiency and effectiveness in detecting evolving XSS attacks and effectively responding to feature drift through temporal and spatial dynamic feature selection across different data types, including payloads and JS files, and utilizing both numerical and word embedding representations, has laid a solid foundation for other researchers who work in such related fields to extend the thesis work for further implementations. Several promising avenues for future work emerge as follows:

1. Implementation Across other Domains:

The success of our proposed framework suggests its potential for implementation across diverse domains, such as IoT (Internet of Things) and Industry 4.0, where cyber threats pose significant challenges. Integrating our framework into these domains could yield valuable insights into its adaptability and performance under different contexts.

2. Extension to Other Web Application Attacks:

Expanding the scope of the framework to cover other types of web application attacks is a logical step. For instance, extending the framework to handle SQL injection attacks, cross-site request forgery (CSRF), and other common web application vulnerabilities would provide a comprehensive security solution for a broader range of threats.

3. Integration with Existing Systems:

The evolving nature of cyber threats calls for dynamic update mechanisms. Investigating methods to dynamically update the framework's knowledge base and models in response to emerging attack patterns could enhance its ability to stay up-to-date with the ever-changing threat landscape. Thus, integrating our framework with real-time detection systems would enable immediate response to emerging threats, minimizing the window of vulnerability. This could involve developing adaptive models that can incorporate new data and patterns without extensive retraining.

The research presented in this thesis has made significant contributions to the field of XSS attack detection. The proposed DA-XSS-AD framework, with its innovative approaches to dynamic feature selection, adaptability to evolving data, and capacity represents a substantial advancement in the domain. While acknowledging the limitations, the future directions outlined provide a promising pathway for continued research and development. The findings of this study not only contribute to the academic community but also have the potential to impact the broader cybersecurity landscape, enhancing the detection and prevention of XSS attacks.

REFERENCES

- Abaimov, S., & Bianchi, G. (2019). CODDLE: Code-Injection Detection with Deep Learning. *IEEE Access*, 7, 128617–128627. <https://doi.org/10.1109/ACCESS.2019.2939870>
- Abu Al-Haija, Q. (2023). Cost-effective detection system of cross-site scripting attacks using hybrid learning approach. *Results in Engineering*, 19(June), 101266. <https://doi.org/10.1016/j.rineng.2023.101266>
- Akaishi, S., Uda, R., & IEEE. (2019). Classification of XSS Attacks by Machine Learning with Frequency of Appearance and Co-occurrence. In *2019 53rd Annual Conference on Information Sciences and Systems, CISS 2019*. IEEE. <https://doi.org/10.1109/CISS.2019.8693047>
- Alazab, A., Khraisat, A., Alazab, M., & Singh, S. (2022). Detection of Obfuscated Malicious JavaScript Code. *Future Internet*, 14(8), 217. <https://doi.org/10.3390/fi14080217>
- Alhamyani, R., & Alshammari, M. (2024). Machine Learning-Driven Detection of Cross-Site Scripting Attacks. *Information*, 15(7), 420.
- Alon, U., Zilberstein, M., Levy, O., & Yahav, E. (2019). Code2Vec: Learning Distributed Representations of Code. *Proceedings of the ACM on Programming Languages*, 3(POPL), 1–29. <https://doi.org/10.1145/3290353>
- Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. In *Journal of Big Data* (Vol. 8, Issue 1). Springer International Publishing. <https://doi.org/10.1186/s40537-021-00444-8>
- Andrew, A. M. (1998). Reinforcement Learning: An Introduction. *Kybernetes*, 27(9), 1093–1096. <https://doi.org/10.1108/k.1998.27.9.1093.3>
- Arulkumaran, K., Deisenroth, Marc Peter Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34, 26–38.
- Bada, M., Sasse, A. M., & Nurse, J. R. C. (2019). Cyber Security Awareness Campaigns: Why do they fail to change behaviour? *International Conference on Cyber Security for Sustainable Society, July*, 38. <http://arxiv.org/abs/1901.02672>
- Bai, S., Zico Kolter, J., & Koltun, V. (2018). *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*. <http://github.com/locuslab/TCN>.
- Barddal, J. P., Enembreck, F., Gomes, H. M., Bifet, A., & Pfahringer, B. (2019). Boosting decision stumps for dynamic feature selection on data streams. *Information Systems*, 83, 13–29. <https://doi.org/10.1016/j.is.2019.02.003>

- Barddal, J. P., Gomes, H. M., Enembreck, F., & Pfahringer, B. (2017). A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *Journal of Systems and Software*, 127, 278–294. <https://doi.org/10.1016/j.jss.2016.07.005>
- Barddal, J. P., Gomes, H. M., Enembreck, F., Pfahringer, B., & Bifet, A. (2016). On dynamic feature weighting for feature drifting data streams. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9852 LNAI, 129–144. https://doi.org/10.1007/978-3-319-46227-1_9
- Barto, R. S. S. A. A. G. (2018). *Reinforcement Learning, second edition: An Introduction*. MIT press.
- Bifet, A., Holmes, G., Pfahringer, B., & Gavaldà, R. (2009). Improving adaptive bagging methods for evolving data streams. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5828 LNAI, 23–37. https://doi.org/10.1007/978-3-642-05224-8_4
- Calzavara, S., Urban, T., Tatang, D., Steffens, M., & Stock, B. (2021). Reining in the Web's Inconsistencies with Site Policy. *Proceedings 2021 Network and Distributed System Security Symposium, February*. <https://doi.org/10.14722/ndss.2021.23091>
- Cavero, S., Pardo, E. G., Glover, F., & Martí, R. (2024). Strategic oscillation tabu search for improved hierarchical graph drawing. *Expert Systems with Applications*, 243(November 2023), 122668. <https://doi.org/10.1016/j.eswa.2023.122668>
- CERT CC. (2001). 2001 CERT Advisories. *Whitepaper*, 23. https://resources.sei.cmu.edu/asset_files/WhitePaper/2001_019_001_496192.pdf
- Cervantes, J., Garcia-Lamont, F., Rodríguez-Mazahua, L., & Lopez, A. (2020). A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408, 189–215. <https://doi.org/10.1016/j.neucom.2019.10.118>
- Chandrashekar, G., & Sahin, F. (2014). A survey on feature selection methods. *Computers and Electrical Engineering*, 40(1), 16–28. <https://doi.org/10.1016/j.compeleceng.2013.11.024>
- Chaudhary, P., Gupta, B. B., Chang, X., Nedjah, N., & Chui, K. T. (2021). Enhancing big data security through integrating XSS scanner into fog nodes for SMEs gain. *Technological Forecasting and Social Change*, 168, 120754. <https://doi.org/https://doi.org/10.1016/j.techfore.2021.120754>
- Che, J., Yang, Y., Li, L., Bai, X., Zhang, S., & Deng, C. (2017). Maximum relevance minimum common redundancy feature selection for nonlinear data. *Information Sciences*, 409–410, 68–86. <https://doi.org/10.1016/j.ins.2017.05.013>

- Chen, H.-C., Nshimiyimana, A., Damarjati, C., & Chang, P.-H. (2021). Detection and Prevention of Cross-site Scripting Attack with Combined Approaches. *2021 International Conference on Electronics, Information, and Communication (ICEIC)*, 1–4. <https://doi.org/10.1109/ICEIC51217.2021.9369796>
- Chen, L., Tang, C., He, J., Zhao, H., Lan, X., & Li, T. (2022). XSS adversarial example attacks based on deep reinforcement learning. *Computers and Security*, 120, 102831. <https://doi.org/10.1016/j.cose.2022.102831>
- Cheng, Y., Komatsu, K., Sato, M., & Kobayashi, H. (2021). A Deep Reinforcement Learning Based Feature Selector. *Communications in Computer and Information Science*, 1362, 378–389. https://doi.org/10.1007/978-981-16-0010-4_33
- Cheng, Z., Cui, B., Qi, T., Yang, W., & Fu, J. (2021). An Improved Feature Extraction Approach for Web Anomaly Detection Based on Semantic Structure. *Security and Communication Networks*, 2021. <https://doi.org/10.1155/2021/6661124>
- Choi, H., Hong, S., Cho, S., Kim, Y. G., & IEEE. (2017). HXD: Hybrid XSS Detection by using a Headless Browser. In *2017 4TH INTERNATIONAL CONFERENCE ON COMPUTER APPLICATIONS AND INFORMATION PROCESSING TECHNOLOGY (CAIPT)* (pp. 388–391).
- Christensen, A. S., Møller, A., & Schwartzbach, M. I. (2003). Precise analysis of string expressions. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2694(February), 1–18. https://doi.org/10.1007/3-540-44898-5_1
- Chromium. (2019). *XSS Auditor*. <https://www.chromium.org/developers/design-documents/xss-auditor/>
- Cieslak, D. A., & Chawla, N. V. (2008). Learning decision trees for unbalanced data. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5211 LNAI(PART 1), 241–256. https://doi.org/10.1007/978-3-540-87479-9_34
- Cooper, K. (2020). *Alexa*. <https://www.alexa.com/>
- Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297. <https://doi.org/10.1007/BF00994018>
- Covert, I., Qiu, W., Lu, M., Kim, N., & White, N. (2023). Learning to Maximize Mutual Information for Dynamic Feature Selection. *International Conference on Machine Learning*, 6424–6447.
- Cui, B., He, S., Shi, P., & Yao, X. (2018). Malicious URL detection with feature extraction based on machine learning. *International Journal of High Performance Computing and Networking*, 12(2), 166–178. <https://doi.org/10.1504/ijhpcn.2018.094367>

- Damodaran, A., Troia, F. Di, Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, 13(1), 1–12. <https://doi.org/10.1007/s11416-015-0261-z>
- Dong, Y., Wang, R., & He, J. (2019). Real-time network intrusion detection system based on deep learning. *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS, 2019-Octob*, 1–4. <https://doi.org/10.1109/ICSESS47205.2019.9040718>
- Duoergun. (2017). *GitHub*. <https://github.com/duoergun0729/1book/tree/master/data>
- Fahy, C., & Yang, S. (2019). Dynamic Feature Selection for Clustering High Dimensional Data Streams. *IEEE Access*, 7, 127128–127140. <https://doi.org/10.1109/ACCESS.2019.2932308>
- Fan, W., Liu, K., Liu, H., Ge, Y., Xiong, H., & Fu, Y. (2020). *Interactive Reinforcement Learning for Feature Selection with Decision Tree in the Loop*. 1–12. <https://doi.org/10.1109/tkde.2021.3102120>
- Fan, W., Liu, K., Liu, H., Hariri, A., Dou, D., & Fu, Y. (2021). AutoGFS: Automated Group-based Feature Selection via Interactive Reinforcement Learning. *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*, 342–350. <https://doi.org/10.1137/1.9781611976700.39>
- Fang, Y., Huang, C., Liu, L., & Xue, M. (2018). Research on malicious JavaScript detection technology based on LSTM. *IEEE Access*, 6, 59118–59125. <https://doi.org/10.1109/ACCESS.2018.2874098>
- Fang, Y., Huang, C., Su, Y., & Qiu, Y. (2020). Detecting malicious JavaScript code based on semantic analysis. *Computers & Security*, 93, 101764. <https://doi.org/https://doi.org/10.1016/j.cose.2020.101764>
- Fang, Y., Huang, C., Xu, Y. J., & Li, Y. (2019). RLXSS: Optimizing XSS Detection Model to Defend Against Adversarial Attacks Based on Reinforcement Learning. *FUTURE INTERNET*, 11(8). <https://doi.org/10.3390/fi11080177>
- Fang, Y., Huang, C., Zeng, M., Zhao, Z., & Huang, C. (2022). JStrong: Malicious JavaScript detection based on code semantic representation and graph neural network. *Computers and Security*, 118, 102715. <https://doi.org/10.1016/j.cose.2022.102715>
- Fang, Y., Li, Y., Liu, L., & Huang, C. (2018). DeepXSS: Cross site scripting detection based on deep learning. *ACM International Conference Proceeding Series*, 47–51. <https://doi.org/10.1145/3194452.3194469>
- Fang, Y., Xu, Y., Jia, P., & Huang, C. (2020). Providing Email Privacy by Preventing Webmail from Loading Malicious XSS Payloads. *Applied Sciences*, 10(13), 4425. <https://doi.org/10.3390/app10134425>

- Fang, Z., Wang, J., Geng, J., & Kan, X. (2019). Feature Selection for Malware Detection Based on Reinforcement Learning. *IEEE Access*, 7, 176177–176187. <https://doi.org/10.1109/ACCESS.2019.2957429>
- Fawaz, M. (2022). *XSS-dataset*. <https://github.com/fawaz2015/XSS-Dataset>
- Ferone, A., & Maratea, A. (2021). Adaptive quick reduct for feature drift detection. *Algorithms*, 14(2). <https://doi.org/10.3390/a14020058>
- Field, A. (2009). *Discovering Statistics Using SPSS*. SAGE Publications. <https://books.google.com.my/books?id=4mEOw7xa3z8C>
- Frank, E., Hall, M. A. ., Pal, C. J. ., & Witten, I. H. (2017). *Data mining: practical machine learning tools and techniques* (Fourth edi). Morgan Kaufmann Publishers Inc.
- Fred Glover. (1989). Tabu Search - Part I. *Orsa Journal on Computing*, 1(3), 190–206.
- Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., & Bouchachia, A. (2014). A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4), 1–37. <https://doi.org/10.1145/2523813>
- Gao, C., Yan, J., Zhou, S., Varshney, P. K., & Liu, H. (2019). Long short-term memory-based deep recurrent neural networks for target tracking. *Information Sciences*, 502, 279–296. <https://doi.org/10.1016/j.ins.2019.06.039>
- Garcia, C. M., Abilio, R. S., Koerich, A. L., Britto, A. de S., & Barddal, J. P. (2023). *Concept Drift Adaptation in Text Stream Mining Settings: A Comprehensive Review*. <http://arxiv.org/abs/2312.02901>
- Garcia, C. M., Koerich, A. L., Britto, A. de S., & Barddal, J. P. (2024). *Methods for Generating Drift in Text Streams*. <http://arxiv.org/abs/2403.12328>
- Geetha, R., & Thilagam, T. (2021). A Review on the Effectiveness of Machine Learning and Deep Learning Algorithms for Cyber Security. *Archives of Computational Methods in Engineering*, 28(4), 2861–2879. <https://doi.org/10.1007/s11831-020-09478-2>
- Ghaleb, F. A., Alsaedi, M., Saeed, F., Ahmad, J., & Alasli, M. (2022). Cyber Threat Intelligence-Based Malicious URL Detection Model Using Ensemble Learning. *Sensors*, 22(9). <https://doi.org/10.3390/s22093373>
- Giorgio Maone. (2005). *NoScript*. <https://noscript.net/>
- Glover, F., & Laguna, M. (1998). Tabu Search. *Handbook of Combinatorial Optimization*, 2093–2229. https://doi.org/10.1007/978-1-4613-0303-9_33
- Gong, L., & Ji, R. (2018). *What Does a TextCNN Learn?* 1–3. <http://arxiv.org/abs/1801.06287>

- González-Estrada, E., & Cosmes, W. (2019). Shapiro–Wilk test for skew normal distributions based on data transformations. *Journal of Statistical Computation and Simulation*, 89(17), 3258–3272. <https://doi.org/10.1080/00949655.2019.1658763>
- Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18(5–6), 602–610. <https://doi.org/10.1016/j.neunet.2005.06.042>
- Gronauer, S., & Diepold, K. (2022). Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2), 895–943. <https://doi.org/10.1007/s10462-021-09996-w>
- Guo, Z. (2021). Malicious JavaScript Detection Method Based on Multilayer Perceptron. In *Advances in Intelligent Systems and Computing* (Vol. 1282). https://doi.org/10.1007/978-3-030-62743-0_74
- Gupta, C., Singh, R. K., & Mohapatra, A. K. (2022). GeneMiner: A Classification Approach for Detection of XSS Attacks on Web Services. *Computational Intelligence and Neuroscience*, 2022(1), 1–12. <https://doi.org/10.1155/2022/3675821>
- Gupta, S., & Gupta, B. B. (2016). JS-SAN: defense mechanism for HTML5-based web applications against javascript code injection vulnerabilities. *Security and Communication Networks*, 9(11), 1477–1495. <https://doi.org/10.1002/sec.1433>
- Gupta, S., & Gupta, B. B. (2017). Cross-Site Scripting (XSS) attacks and defense mechanisms: classification and state-of-the-art. *International Journal of Systems Assurance Engineering and Management*, 8(s1), 512–530. <https://doi.org/10.1007/s13198-015-0376-0>
- Gupta, S., & Gupta, B. B. (2019). Evaluation and monitoring of XSS defensive solutions: a survey, open research issues and future directions. *JOURNAL OF AMBIENT INTELLIGENCE AND HUMANIZED COMPUTING*, 10(11), 4377–4405. <https://doi.org/10.1007/s12652-018-1118-3>
- Gupta, S., Gupta, B. B., & Chaudhary, P. (2020). Nested context-aware sanitisation and feature injection in clustered templates of JavaScript worms on the cloud-based OSN. *International Journal of Information and Computer Security*, 12(2/3), 147. <https://doi.org/10.1504/IJICS.2020.105153>
- Hamdi, A., Shaban, K., Erradi, A., Mohamed, A., Rumi, S. K., & Salim, F. D. (2022). Spatiotemporal data mining: a survey on challenges and open problems. In *Artificial Intelligence Review* (Vol. 55, Issue 2). Springer Netherlands. <https://doi.org/10.1007/s10462-021-09994-y>
- Hannousse, A., Yahiouche, S., & Nait-Hamoud, M. C. (2024). Twenty-two years since revealing cross-site scripting attacks: A systematic mapping and a comprehensive survey. *Computer Science Review*, 52(April), 100634. <https://doi.org/10.1016/j.cosrev.2024.100634>

- He, X., Xu, L., & Cha, C. (2018). Malicious JavaScript Code Detection Based on Hybrid Analysis. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2018-Decem*, 365–374. <https://doi.org/10.1109/APSEC.2018.00051>
- Heiderich, M., Schwenk, J., Frosch, T., Magazinius, J., & Yang, E. Z. (2013). mXSS attacks: Attacking well-secured web-applications by using innerHTML mutations. *Proceedings of the ACM Conference on Computer and Communications Security*, 777–788. <https://doi.org/10.1145/2508859.2516723>
- Hidayat, A. (2018). *Esprima Documentation*. <https://esprima.org/doc/index.html>
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780. <https://doi.org/10.1162/neco.1997.9.8.1735>
- Hosseinzadeh, M., Rahmani, A. M., Vo, B., Bidaki, M., Masdari, M., & Zangakani, M. (2021). Improving security using SVM-based anomaly detection: issues and challenges. *Soft Computing*, 25(4), 3195–3223. <https://doi.org/10.1007/s00500-020-05373-x>
- Hu, T., Xu, C., Zhang, S., Tao, S., & Li, L. (2023). Cross-site scripting detection with two-channel feature fusion embedded in self-attention mechanism. *Computers and Security*, 124, 102990. <https://doi.org/10.1016/j.cose.2022.102990>
- Huang, J., Dang, J., Borchert, G. M., Eilbeck, K., Zhang, H., Xiong, M., Jiang, W., Wu, H., Blake, J. A., Natale, D. A., & Tan, M. (2014). OMIT: Dynamic, semi-automated ontology development for the microRNA domain. *PLoS ONE*, 9(7). <https://doi.org/10.1371/journal.pone.0100855>
- Huang, Y., Li, T., Zhang, L., Li, B., & Liu, X. (2021). JSContana: Malicious JavaScript detection using adaptable context analysis and key feature extraction. *Computers and Security*, 104, 102218. <https://doi.org/10.1016/j.cose.2021.102218>
- Huang, Z., Xu, W., & Yu, K. (2015). *Bidirectional LSTM-CRF Models for Sequence Tagging*. <http://arxiv.org/abs/1508.01991>
- Humayun, M., Niazi, M., Jhanjhi, N., Alshayeb, M., & Mahmood, S. (2020). Cyber Security Threats and Vulnerabilities: A Systematic Mapping Study. *Arabian Journal for Science and Engineering*, 45(4), 3171–3189. <https://doi.org/10.1007/s13369-019-04319-2>
- Hydara, I., Sultan, A. B. M., Zulzalil, H., & Admodisastro, N. (2015). Current state of research on cross-site scripting (XSS) - A systematic literature review. *Information and Software Technology*, 58, 170–186. <https://doi.org/10.1016/j.infsof.2014.07.010>
- HynekPetrak. (2017). *Javascript Malware Collection*. <https://github.com/HynekPetrak/javascript-malware-collection>
- IramTariq. (2020). *XSS-attack-detection*. GitHub. <https://github.com/IramTariq/XSS-attack-detection>

- İsmail Taşdelen. (2021). *XSS Payload Box*. <https://github.com/payloadbox/xss-payload-list/>
- Jim Manico, R. Rs. H. (2018). *XSS (Cross Site Scripting) Prevention Cheat Sheet - OWASP*. <https://owasp.org/www-community/xss-filter-evasion-cheatsheet>
- Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4(19), 237–285. <https://doi.org/10.1613/jair.301>
- Kaur, J., Garg, U., & Bathla, G. (2023). Detection of cross-site scripting (XSS) attacks using machine learning techniques: a review. *Artificial Intelligence Review*, 56(11), 12725–12769. <https://doi.org/10.1007/s10462-023-10433-3>
- Kim, M., Bae, J., Wang, B., Ko, H., & Lim, J. S. (2022). Feature Selection Method Using Multi-Agent Reinforcement Learning Based on Guide Agents. *Sensors*, 23(1), 98. <https://doi.org/10.3390/s23010098>
- Kim, Y. (2014). Convolutional neural networks for sentence classification. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 1746–1751. <https://doi.org/10.3115/v1/d14-1181>
- Kotowicz, K. (2020, March 25). *Prevent DOM-based cross-site scripting vulnerabilities with Trusted Types*. <https://web.dev/trusted-types/>
- Kronjee, J., Hommersom, A., & Vranken, H. (2018). Discovering software vulnerabilities using data-flow analysis and machine learning. *ACM International Conference Proceeding Series*. <https://doi.org/10.1145/3230833.3230856>
- Kumar, J., Santhanavijayan, A., & Rajendran, B. (2022). Cross Site Scripting Attacks Classification using Convolutional Neural Network. *2022 International Conference on Computer Communication and Informatics, ICCCI 2022*. <https://doi.org/10.1109/ICCCI54379.2022.9740836>
- Kuppa, K., Dayal, A., Gupta, S., Dua, A., Chaudhary, P., & Rathore, S. (2022). ConvXSS: A deep learning-based smart ICT framework against code injection attacks for HTML5 web applications in sustainable smart city infrastructure. *Sustainable Cities and Society*, 80, 103765. <https://doi.org/10.1016/j.scs.2022.103765>
- Lee, S., Wi, S., & Son, S. (2022). Link: Black-Box Detection of Cross-Site Scripting Vulnerabilities Using Reinforcement Learning. *WWW 2022 - Proceedings of the ACM Web Conference 2022, April*, 743–754. <https://doi.org/10.1145/3485447.3512234>
- Lei, L., Chen, M., He, C., & Li, D. (2020). XSS Detection Technology Based on LSTM-Attention. *2020 5th International Conference on Control, Robotics and Cybernetics (CRC)*, 175–180. <https://doi.org/10.1109/CRC51253.2020.9253484>

- Lekies, S. (2023). *Client-Side Cross-Site Scripting: Exploitation, Detection, Mitigation, and Prevention*. Technischen Universität Carolo-Wilhelmina zu Braunschweig.
- Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., & Liu, H. (2017). Feature selection: A data perspective. *ACM Computing Surveys*, 50(6). <https://doi.org/10.1145/3136625>
- Li, X., Ma, W., Zhou, Z., & Xu, C. (2020). XSS Attack Detection Model Based on Semi-supervised Learning Algorithm with Weighted Neighbor Purity. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics): Vol. 12338 LNCS* (pp. 198–213). Springer International Publishing. https://doi.org/10.1007/978-3-030-61746-2_15
- Li, X., Wang, T., Zhang, W., Niu, X., Zhang, T., Zhao, T., Wang, Y., & Wang, Y. (2023). An LSTM based cross-site scripting attack detection scheme for Cloud Computing environments. *Journal of Cloud Computing*, 12(1). <https://doi.org/10.1186/s13677-023-00483-x>
- Li, X., Yao, J., Ren, J., & Wang, L. (2021). A New Feature Selection Algorithm Based on Deep Q-Network. *Chinese Control Conference, CCC, 2021-July*, 7100–7105. <https://doi.org/10.23919/CCC52363.2021.9550745>
- Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. *Eleventh International Conference on International Conference on Machine Learning*, 157–163.
- Liu, J., & Xu, Y. (2022). T-Friedman Test: A New Statistical Test for Multiple Comparison with an Adjustable Conservativeness Measure. *International Journal of Computational Intelligence Systems*, 15(1), 1–19. <https://doi.org/10.1007/s44196-022-00083-8>
- Liu, K., Fu, Y., Wu, L., Li, X., Aggarwal, C., & Xiong, H. (2021). Automated Feature Selection: A Reinforcement Learning Perspective. *IEEE Transactions on Knowledge and Data Engineering*, 4347(c). <https://doi.org/10.1109/TKDE.2021.3115477>
- Liu, M., Zhang, B. Y., Chen, W. B., & Zhang, X. L. (2019). A Survey of Exploitation and Detection Methods of XSS Vulnerabilities. *IEEE ACCESS*, 7, 182004–182016. <https://doi.org/10.1109/ACCESS.2019.2960449>
- Liu, Z., Fang, Y., Huang, C., & Han, J. (2021). GraphXSS: An Efficient XSS Payload Detection Approach Based on Graph Convolutional Network. *Computers & Security*, 102597. <https://doi.org/10.1016/j.cose.2021.102597>
- Liu, Z., Fang, Y., Huang, C., & Xu, Y. (2022). GAXSS: Effective Payload Generation Method to Detect XSS Vulnerabilities Based on Genetic Algorithm. *Security and Communication Networks*, 2022. <https://doi.org/10.1155/2022/2031924>

- Liu, Z., Fang, Y., Huang, C., & Xu, Y. (2023a). MFXSS: An effective XSS vulnerability detection method in JavaScript based on multi-feature model. *Computers and Security*, 124, 103015. <https://doi.org/10.1016/j.cose.2022.103015>
- Liu, Z., Fang, Y., Huang, C., & Xu, Y. (2023b). MFXSS: An effective XSS vulnerability detection method in JavaScript based on multi-feature model. *Computers & Security*, 124, 103015. <https://doi.org/10.1016/j.cose.2022.103015>
- Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., & Zhang, G. (2019). Learning under Concept Drift: A Review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12), 2346–2363. <https://doi.org/10.1109/TKDE.2018.2876857>
- Lu, J., Wei, Z., Qin, Z., Chang, Y., & Zhang, S. (2022). Resolving Cross-Site Scripting Attacks through Fusion Verification and Machine Learning. *Mathematics*, 10(20). <https://doi.org/10.3390/math10203787>
- Lun, Y. Z., D’Innocenzo, A., Malavolta, I., & Di Benedetto, M. D. (2016). *Cyber-Physical Systems Security: a Systematic Mapping Study*. 1–32. <https://doi.org/10.1016/j.jss.2018.12.006>
- Lv, C., Zhang, L., Zeng, F., & Zhang, J. (2019). Adaptive Random Testing for XSS Vulnerability. *Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2019-Decem*, 63–69. <https://doi.org/10.1109/APSEC48747.2019.00018>
- Malviya, V. K., Rai, S., & Gupta, A. (2021). Development of web browser prototype with embedded classification capability for mitigating Cross-Site Scripting attacks. *Applied Soft Computing*, 102, 106873. <https://doi.org/10.1016/j.asoc.2020.106873>
- Malviya, V., Rai, S., & Gupta, A. (2018). Development of a plugin based extensible feature extraction framework. *Proceedings of the ACM Symposium on Applied Computing*, 1840–1847. <https://doi.org/10.1145/3167132.3167328>
- Maurel, H., Vidal, S., & Rezk, T. (2022). Statically identifying XSS using deep learning. *Science of Computer Programming*, 219, 102810. <https://doi.org/10.1016/j.scico.2022.102810>
- McAlaney, J., Benson, V., & Frumkin, L. A. (2018). Psychological and Behavioral Examinations in Cyber Security. *Psychological and Behavioral Examinations in Cyber Security*, 1–334. <https://doi.org/10.4018/978-1-5225-4053-3>
- Melicher, W., Das, A., Sharif, M., Bauer, L., & Jia, L. (2018). *Riding out DOMsday: Towards Detecting and Preventing DOM Cross-Site Scripting*. February. <https://doi.org/10.14722/ndss.2018.23309>
- Melicher, W., & Fung, C. (2021). Towards a Lightweight , Hybrid Approach for Detecting DOM XSS Vulnerabilities with Machine Learning. *International World Wide Web Conference (WWW’21)*. <https://doi.org/10.1145/3442381.3450062>

- Melicher, W., Fung, C., Bauer, L., & Jia, L. (2021). Towards a Lightweight, Hybrid Approach for Detecting DOM XSS Vulnerabilities with Machine Learning. *Proceedings of the Web Conference 2021*, 2684–2695. <https://doi.org/10.1145/3442381.3450062>
- Mereani, F. A., & Howe, J. M. (2018). Detecting Cross-Site Scripting Attacks Using Machine Learning. In *Advances in Intelligent Systems and Computing* (Vol. 723). https://doi.org/10.1007/978-3-319-74690-6_20
- Microsoft. (2008). *IE 8 XSS Filter Architecture / Implementation | MSRC Blog | Microsoft Security Response Center*. <https://msrc.microsoft.com/blog/2008/08/ie-8-xss-filter-architecture-implementation/>
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems*, 3111–3119.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013b). Efficient estimation of word representations in vector space. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, 1–12.
- Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. *ArXiv*, 1–9. <http://arxiv.org/abs/1312.5602>
- Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533. <https://doi.org/10.1038/nature14236>
- Mohammadi, M., Chu, B., & Lipford, H. R. (2017). Detecting cross-site scripting vulnerabilities through automated unit testing. *Proceedings - 2017 IEEE International Conference on Software Quality, Reliability and Security, QRS 2017*, 364–373. <https://doi.org/10.1109/QRS.2017.46>
- Mohammadi, M., Chu, B., & Richter Lipford, H. (2019). Automated repair of cross-site scripting vulnerabilities through unit testing. *Proceedings - 2019 IEEE 30th International Symposium on Software Reliability Engineering Workshops, ISSREW 2019*, 370–377. <https://doi.org/10.1109/ISSREW.2019.00098>
- Mohammadi, S., & Namadchian, A. (2020). Anomaly-based Web Attack Detection: The Application of Deep Neural Network Seq2Seq With Attention Mechanism. *ISECURE-ISC INTERNATIONAL JOURNAL OF INFORMATION SECURITY*, 12(1), 44–54. <https://doi.org/10.22042/ISECURE.2020.199009.479>
- Mokbal, F. M. ., Dan, W., Imran, A., Jiuchuan, L., Akhtar, F., & Xiaoxi, W. (2019). MLPXSS: An Integrated XSS-Based Attack Detection Scheme in Web Applications Using Multilayer Perceptron Technique. *IEEE Access*, 7, 100567–100580. <https://doi.org/10.1109/access.2019.2927417>

- Mokbal, F. M. M., Dan, W., Imran, A., Lin, J. C., Akhtar, F., & Wang, X. X. (2019). MLPXSS: An Integrated XSS-Based Attack Detection Scheme in Web Applications Using Multilayer Perceptron Technique. *IEEE ACCESS*, 7, 100567–100580. <https://doi.org/10.1109/ACCESS.2019.2927417>
- Mokbal, F. M. M., Dan, W., Xiaoxi, W., Wenbin, Z., & Lihua, F. (2021). XGBXSS: An Extreme Gradient Boosting Detection Framework for Cross-Site Scripting Attacks Based on Hybrid Feature Selection Approach and Parameters Optimization. *Journal of Information Security and Applications*, 58(March), 102813. <https://doi.org/10.1016/j.jisa.2021.102813>
- Mokbal, F. M. M., Wang, D., & Wang, X. (2022). Detect Cross-Site Scripting Attacks Using Average Word Embedding and Support Vector Machine. *International Journal of Network Security*, 24(January), 20–28. [https://doi.org/10.6633/IJNS.20220124\(1\).03](https://doi.org/10.6633/IJNS.20220124(1).03)
- Mokbal, F. M. M., Wang, D., Wang, X. X., & Fu, L. H. (2020). Data augmentation-based conditional Wasserstein generative adversarial network-gradient penalty for XSS attack detection system. *PEERJ COMPUTER SCIENCE*, 6, 1–20. <https://doi.org/10.7717/peerj-cs.328>
- MSE. (2000). *Happy 10th birthday Cross-Site Scripting! | Microsoft Learn*. <https://learn.microsoft.com/en-ca/archive/blogs/dross/happy-10th-birthday-cross-site-scripting>
- Nagarjun, P., & Ahamad, S. S. (2020). Ensemble methods to detect XSS attacks. *International Journal of Advanced Computer Science and Applications*, 11(5), 695–700. <https://doi.org/10.14569/IJACSA.2020.0110585>
- Nanfack, G., Temple, P., & Frénay, B. (2022). Constraint Enforcement on Decision Trees: A Survey. *ACM Computing Surveys*, 54(10). <https://doi.org/10.1145/3506734>
- Ndichu, S., Kim, S., Ozawa, S., Misu, T., & Makishima, K. (2019). A machine learning approach to detection of JavaScript-based attacks using AST features and paragraph vectors. *Applied Soft Computing*, 84, 105721. <https://doi.org/10.1016/j.asoc.2019.105721>
- Ngoc, P. M., & Mimura, M. (2021). Oversampling for Detection of Malicious JavaScript in Realistic Environment. *Lecture Notes in Networks and Systems*, 159 LNNS, 176–187. https://doi.org/10.1007/978-3-030-61108-8_17
- Nguyen, H. L., Woon, Y. K., Ng, W. K., & Wan, L. (2012). Heterogeneous ensemble for feature drifts in data streams. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7301 LNAI(PART 2), 1–12. https://doi.org/10.1007/978-3-642-30220-6_1
- Nguyen, T. T., Nguyen, N. D., Nahavandi, S., & Member, S. (2020). *Deep Reinforcement Learning for Multiagent Systems: A Review of Challenges, Solutions, and Applications*. 50(9), 3826–3839.

- Niu, Q., & Li, X. (2020). A High-performance Web Attack Detection Method based on CNN-GRU Model. *Proceedings of 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference, ITNEC 2020, Itnec*, 804–808. <https://doi.org/10.1109/ITNEC48623.2020.9085028>
- Open Bug Bounty. (n.d.). *Coordinated Vulnerability Disclosure*. Retrieved March 28, 2023, from <https://www.openbugbounty.org/>
- Oroojlooy, A., & Hajinezhad, D. (2023). A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, 53(11), 13677–13722. <https://doi.org/10.1007/s10489-022-04105-y>
- OWASP. (2017). *OWASP Top 10-2017*. <https://github.com/OWASP/Top10/issues>
- OWASP. (2021). OWASP Top 10:2021. In *OWASP Top 10- 2021*. <https://owasp.org/Top10/>
- Pan, H., Fang, Y., Huang, C., Guo, W., & Wan, X. (2022). GCNXSS: An Attack Detection Approach for Cross-Site Scripting Based on Graph Convolutional Networks. *KSII Transactions on Internet and Information Systems*, 16(12), 4008–4023. <https://doi.org/10.3837/tiis.2022.12.013>
- Pan, Y., Sun, F., Teng, Z., White, J., Schmidt, D. C., Staples, J., & Krause, L. (2019). Detecting web attacks with end-to-end deep learning. *Journal of Internet Services and Applications*, 10(1). <https://doi.org/10.1186/s13174-019-0115-x>
- Pandey, A. K., & Alsolami, F. (2020). Malware analysis in web application security: An investigation and suggestion. *International Journal of Advanced Computer Science and Applications*, 11(7), 191–201. <https://doi.org/10.14569/IJACSA.2020.0110725>
- Panigrahi, R., Borah, S., Pramanik, M., Bhoi, A. K., Barsocchi, P., Nayak, S. R., & Alnumay, W. (2022). Intrusion detection in cyber–physical environment using hybrid Naïve Bayes—Decision table and multi-objective evolutionary feature selection. *Computer Communications*, 188(February), 133–144. <https://doi.org/10.1016/j.comcom.2022.03.009>
- Paniri, M., Dowlatshahi, M. B., & Nezamabadi-pour, H. (2021). Ant-TD: Ant colony optimization plus temporal difference reinforcement learning for multi-label feature selection. *Swarm and Evolutionary Computation*, 64(December 2019), 100892. <https://doi.org/10.1016/j.swevo.2021.100892>
- Park, C., Won, S., Jin, J., & Ryu, S. (2016). Static analysis of JavaScript web applications in the wild via practical DOM modeling. *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, 552–562. <https://doi.org/10.1109/ASE.2015.27>
- Patel, H. H., & Prajapati, P. (2018). Study and Analysis of Decision Tree Based Classification Algorithms. *International Journal of Computer Sciences and Engineering*, 6(10), 74–78. <https://doi.org/10.26438/ijcse/v6i10.7478>

- Peters, M. E., Ammar, W., Bhagavatula, C., & Power, R. (2017). Semi-supervised sequence tagging with bidirectional language models. *ACL 2017 - 55th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference (Long Papers)*, 1, 1756–1765. <https://doi.org/10.18653/v1/P17-1161>
- Pham, T. H., & Raahemi, B. (2023). Bio-Inspired Feature Selection Algorithms With Their Applications: A Systematic Literature Review. *IEEE Access*, 11(May), 43733–43758. <https://doi.org/10.1109/ACCESS.2023.3272556>
- Phung, N. M., & Mimura, M. (2021). Detection of malicious javascript on an imbalanced dataset. *Internet of Things*, 13, 100357. <https://doi.org/10.1016/j.iot.2021.100357>
- Pietraszek, T., & Berghe, C. Vanden. (2006). Defending against injection attacks through context-sensitive string evaluation. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 3858 LNCS, 124–145. https://doi.org/10.1007/11663812_7
- Potochnik, A., Colombo, M., & Wright, C. (2018). Statistics and Probability. *Recipes for Science, Table 2*, 167–206. <https://doi.org/10.4324/9781315686875-6>
- Q-Success. (n.d.). *Web Technology Surveys*. Retrieved July 26, 2024, from <https://w3techs.com/technologies/details/cp-javascript>
- Qin, Z.-Q., Ma, X.-K., & Wang, Y.-J. (2018). Attentional Payload Anomaly Detector for Web Applications. In L. Cheng, A. C. S. Leung, & S. Ozawa (Eds.), *Neural Information Processing* (pp. 588–599). Springer International Publishing.
- Ray, S. (2019). A Quick Review of Machine Learning Algorithms. *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 35–39. <https://doi.org/10.1109/COMITCon.2019.8862451>
- Ren, K., Zeng, Y., Zhong, Y., Sheng, B., & Zhang, Y. (2023). MAFSIDS: a reinforcement learning-based intrusion detection model for multi-agent feature selection networks. *Journal of Big Data*, 10(1). <https://doi.org/10.1186/s40537-023-00814-4>
- Rodriguez, G. E., Torres, J. G., Flores, P., & Benavides, D. E. (2020). Cross-site scripting (XSS) attacks and mitigation: A survey. *COMPUTER NETWORKS*, 166. <https://doi.org/10.1016/j.comnet.2019.106960>
- Rozi, M. F., Ozawa, S., Ban, T., Kim, S., Takahashi, T., & Inoue, D. (2022). Understanding the Influence of AST-JS for Improving Malicious Webpage Detection. *Applied Sciences (Switzerland)*, 12(24). <https://doi.org/10.3390/app122412916>
- Sahoo, D., Liu, C., & Hoi, S. C. H. (2019). Malicious URL Detection using Machine Learning: A Survey. *ArXiv*, 1(1), 1–37. <http://arxiv.org/abs/1701.07179>

- Sarmah, U., Bhattacharyya, D. K., & Kalita, J. K. (2018). A survey of detection methods for XSS attacks. *Journal of Network and Computer Applications*, 118(April), 113–143. <https://doi.org/10.1016/j.jnca.2018.06.004>
- Sarmah, U., Bhattacharyya, D. K., & Kalita, J. K. (2020). XSSD: A Cross-site Scripting Attack Dataset and its Evaluation. *2020 Third ISEA Conference on Security and Privacy (ISEA-ISAP)*, 21–30. <https://doi.org/10.1109/ISEA-ISAP49340.2020.234995>
- Sato, D. M. V., De Freitas, S. C., Barddal, J. P., & Scalabrin, E. E. (2022). A Survey on Concept Drift in Process Mining. *ACM Computing Surveys*, 54(9), 1–37. <https://doi.org/10.1145/3472752>
- Schubert, E. (2020). HACAM: Hierarchical Agglomerative Clustering around Medoids – and its limitations. *CEUR Workshop Proceedings*, 2993, 191–204.
- SecurityScorecard. (n.d.). *CVE security vulnerability database. Security vulnerabilities, exploits, references and more*. Retrieved July 26, 2024, from <https://www.cvedetails.com/vulnerabilities-by-types.php>
- Shahid, W. Bin, Aslam, B., Abbas, H., Khalid, S. Bin, & Afzal, H. (2022). An enhanced deep learning based framework for web attacks detection, mitigation and attacker profiling. *Journal of Network and Computer Applications*, 198(November 2021), 103270. <https://doi.org/10.1016/j.jnca.2021.103270>
- Singh, A. K. (2020). Malicious and Benign Webpages Dataset. *Data in Brief*, 32, 106304. <https://doi.org/10.1016/j.dib.2020.106304>
- Song, X. Y., Chen, C., Cui, B. J., & Fu, J. S. (2020). Malicious JavaScript Detection Based on Bidirectional LSTM Model. *APPLIED SCIENCES-BASEL*, 10(10). <https://doi.org/10.3390/app10103440>
- Song, X., Zhang, R., Dong, Q., & Cui, B. (2023). Grey-Box Fuzzing Based on Reinforcement Learning for XSS Vulnerabilities. *Applied Sciences (Switzerland)*, 13(4). <https://doi.org/10.3390/app13042482>
- Staicu, C. A., Schoepe, D., Balliu, M., Pradel, M., & Sabelfeld, A. (2019). An empirical study of information flows in real-world Javascript. *Proceedings of the ACM Conference on Computer and Communications Security*, 45–59. <https://doi.org/10.1145/3338504.3357339>
- Stamm, S., Sterne, B., & Markham, G. (2010). Reining in the web with content security policy. *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, 1, 921–929. <https://doi.org/10.1145/1772690.1772784>
- Stiawan, D., Bardadi, A., Afifah, N., Melinda, L., Heryanto, A., Wanda Septian, T., Yazid Idris, M., Much Ibnu Subroto, I., Lukman, & Budiarto, R. (2023). An Improved LSTM-PCA Ensemble Classifier for SQL Injection and XSS Attack Detection. *Computer Systems Science and Engineering*, 46(2), 1759–1774. <https://doi.org/10.32604/csse.2023.034047>

- Stock, B., Johns, M., Steffens, M., & Backes, M. (2017). How the web tangled itself: Uncovering the history of client-side web (in)security. *Proceedings of the 26th USENIX Security Symposium*, 971–987.
- Stock, B., Lekies, S., Mueller, T., Spiegel, P., & Johns, M. (2014). Precise client-side protection against DOM-based Cross-Site Scripting. *Proceedings of the 23rd USENIX Security Symposium*, 655–670.
- Sun, N., Zhang, J., Rimba, P., Gao, S., Zhang, L. Y., & Xiang, Y. (2019). Data-Driven Cybersecurity Incident Prediction: A Survey. *IEEE Communications Surveys and Tutorials*, 21(2), 1744–1772. <https://doi.org/10.1109/COMST.2018.2885561>
- Tahmasebi, N., & Risse, T. (2011). Data Mining. In *Mining of Massive Datasets: Vol. 10450 LNCS* (pp. 1–17). Cambridge University Press. <https://doi.org/10.1017/CBO9781139058452.002>
- Tama, B. A., Nkenyereye, L., Islam, S. M. R., & Kwak, K.-S. (2020). An Enhanced Anomaly Detection in Web Traffic Using a Stack of Classifier Ensemble. *IEEE Access*, 8, 24120–24134. <https://doi.org/10.1109/ACCESS.2020.2969428>
- Tariq, I., Sindhu, M. A. M. A. M. A., Abbasi, R. A. R. A., Khattak, A. S. A. S. A. S. A. S., Maqbool, O., & Siddiqui, G. F. G. F. G. F. (2021). Resolving cross-site scripting attacks through genetic algorithm and reinforcement learning. *Expert Systems with Applications*, 168(August 2020), 114386. <https://doi.org/10.1016/j.eswa.2020.114386>
- Tekerek, A. (2021). A novel architecture for web-based attack detection using convolutional neural network. *Computers and Security*, 100. <https://doi.org/10.1016/j.cose.2020.102096>
- The Directory of the Web*. (n.d.). [Http://Www.Dmoz.Org](http://www.Dmoz.Org).
- Tokuda, E. K., Comin, C. H., & Costa, L. da F. (2022). Revisiting agglomerative clustering. *Physica A: Statistical Mechanics and Its Applications*, 585, 126433. <https://doi.org/10.1016/j.physa.2021.126433>
- Usha, G., Kannimuthu, S., Mahendiran, P. D., Shanker, A. K., & Venugopal, D. (2020). Static analysis method for detecting cross site scripting vulnerabilities. *International Journal of Information and Computer Security*, 13(1), 32–47. <https://doi.org/10.1504/IJICS.2020.108123>
- Villa-Blanco, C., Bielza, C., & Larrañaga, P. (2023). Feature subset selection for data and feature streams: a review. In *Artificial Intelligence Review* (Issue 0123456789). Springer Netherlands. <https://doi.org/10.1007/s10462-023-10546-9>
- Vladimir Ivanov. (2016). Web application firewalls: Attacking detection logic mechanisms. In *black hat USA*.
- Von Solms, R., & Van Niekerk, J. (2013). From information security to cyber security. *Computers and Security*, 38, 97–102. <https://doi.org/10.1016/j.cose.2013.04.004>

- VX Vault. (2017). *Malicious URL* . <http://vxvault.net/ViriList.php>
- Wang, Q., Yang, H., Wu, G., Choo, K. R., Zhang, Z., Miao, G., & Ren, Y. (2022). Black-box adversarial attacks on XSS attack detection model. *Computers & Security*, 113, 102554. <https://doi.org/https://doi.org/10.1016/j.cose.2021.102554>
- Wang, R., Xu, G. Q., Zeng, X. J., Li, X. H., & Feng, Z. Y. (2018a). TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting. *JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING*, 118, 100–106. <https://doi.org/10.1016/j.jpdc.2017.07.006>
- Wang, R., Xu, G., Zeng, X., Li, X., & Feng, Z. (2018b). TT-XSS: A novel taint tracking based dynamic detection framework for DOM Cross-Site Scripting. *Journal of Parallel and Distributed Computing*, 118, 100–106. <https://doi.org/10.1016/j.jpdc.2017.07.006>
- Wang, S., Cao, J., & Yu, P. S. (2022). Deep Learning for Spatio-Temporal Data Mining: A Survey. *IEEE Transactions on Knowledge and Data Engineering*, 34(8), 3681–3700. <https://doi.org/10.1109/TKDE.2020.3025580>
- Wang, X., Wang, H., & Wu, D. (2022). Dynamic feature weighting for data streams with distribution-based log-likelihood divergence. *Engineering Applications of Artificial Intelligence*, 107(October 2021), 104509. <https://doi.org/10.1016/j.engappai.2021.104509>
- Wang, Y., Hou, Y., Che, W., & Liu, T. (2020). From static to dynamic word representations: a survey. *International Journal of Machine Learning and Cybernetics*, 11(7), 1611–1630. <https://doi.org/10.1007/s13042-020-01069-8>
- Ward, J. H. (1963). Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301), 236–244. <https://doi.org/10.1080/01621459.1963.10500845>
- Wenzel, F., Galy-Fajou, T., Deutsch, M., & Kloft, M. (2017). Bayesian Nonlinear Support Vector Machines for Big Data. *Lecture Notes in Computer Science*, 10534, 307–322. https://doi.org/10.1007/978-3-319-71249-9_19
- Wu, Y., Li, M., Wang, J., Fang, Z., Zeng, Q., Yang, T., & Cheng, L. (2022). Droidrl: Reinforcement Learning Driven Feature Selection for Android Malware Detection. *SSRN Electronic Journal*, 1–18. <https://doi.org/10.2139/ssrn.4067267>
- XSSed Archive. (2015). *XSSed | Cross Site Scripting (XSS) attacks information and archive*. <http://xssed.com/>
- Xu, H., Kotov, A., Dong, M., Carcone, A. I., Zhu, D., & Naar-King, S. (2016). Text classification with topic-based word embedding and Convolutional Neural Networks. *ACM-BCB 2016 - 7th ACM Conference on Bioinformatics, Computational Biology, and Health Informatics*, 88–97. <https://doi.org/10.1145/2975167.2975176>

- Xu, R., Li, M., Yang, Z., Yang, L., Qiao, K., & Shang, Z. (2021). Dynamic feature selection algorithm based on Q-learning mechanism. *Applied Intelligence*. <https://doi.org/10.1007/s10489-021-02257-x>
- Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. *Insights into Imaging*, 9(4), 611–629. <https://doi.org/10.1007/s13244-018-0639-9>
- Yan, H., Feng, L., Yu, Y., Liao, W., Feng, L., Zhang, J., Liu, D., Zou, Y., Liu, C., Qu, L., & Zhang, X. (2022). Cross-site scripting attack detection based on a modified convolution neural network. *Frontiers in Computational Neuroscience*, 16(August), 1–13. <https://doi.org/10.3389/fncom.2022.981739>
- Yan, R., Xiao, X., Hu, G., Peng, S., & Jiang, Y. (2018). New deep learning method to detect code injection attacks on hybrid applications. *Journal of Systems and Software*, 137, 67–77. <https://doi.org/10.1016/j.jss.2017.11.001>
- Yang, J., Zhou, M., & Cui, B. (2020). MLAB-BiLSTM: Online Web Attack Detection Via Attention-Based Deep Neural Networks. *Communications in Computer and Information Science*, 1268 CCIS, 482–492. https://doi.org/10.1007/978-981-15-9129-7_33
- Yang, W., Zuo, W., & Cui, B. (2019). Detecting Malicious URLs via a Keyword-Based Convolutional Gated-Recurrent-Unit Neural Network. *IEEE Access*, 7, 29891–29900. <https://doi.org/10.1109/ACCESS.2019.2895751>
- Yasuda, T., Bateni, M., Chen, L., Fahrbach, M., Fu, G., & Mirrokni, V. (2022). *Sequential Attention for Feature Selection*. 1–21. <http://arxiv.org/abs/2209.14881>
- Yu, Y., Yan, H., Guan, H., & Zhou, H. (2018). *DeepHTTP: Semantics-Structure Model with Attention for Anomalous HTTP Traffic Detection and Pattern Mining*. <http://arxiv.org/abs/1810.12751>
- Zhang, G., Nie, Y., & Wang, X. (2019). CNNPAYL: An intrusion detection system of cross-site script detection. *ACM International Conference Proceeding Series, Part F1481*, 477–483. <https://doi.org/10.1145/3318299.3318302>
- Zhang, X. Q., Zhou, Y., Pei, S. W., Zhuge, J. J., & Chen, J. H. (2020). Adversarial Examples Detection for XSS Attacks Based on Generative Adversarial Networks. *IEEE ACCESS*, 8, 10989–10996. <https://doi.org/10.1109/ACCESS.2020.2965184>
- Zhang, X., Zhou, Y., Pei, S., Zhuge, J., & Chen, J. (2020). Adversarial Examples Detection for XSS Attacks Based on Generative Adversarial Networks. *IEEE Access*, 8, 10989–10996. <https://doi.org/10.1109/ACCESS.2020.2965184>
- Zhang, Y., Wu, H., & Cheng, L. (2012). Some new deformation formulas about variance and covariance. *Proceedings of 2012 International Conference on Modelling, Identification and Control, ICMIC 2012*, 987–992.

Zhang, Z., Wan, L., Chu, K., Li, S., Wei, H., & Tang, L. (2022). JACNet: Application of adaptive code length network in JavaScript malicious code detection. *PLoS ONE*, 17(12 October), 1–27. <https://doi.org/10.1371/journal.pone.0277891>

Zhou, Y., & Wang, P. (2019). An ensemble learning approach for XSS attack detection with domain knowledge and threat intelligence. *Computers and Security*, 82, 261–269. <https://doi.org/10.1016/j.cose.2018.12.016>

Zhou, Z., Liu, G., & Tang, Y. (2023). Multi-Agent Reinforcement Learning: Methods, Applications, Visionary Prospects, and Challenges. *ArXiv Preprint*, 1(1), 1–43. <http://arxiv.org/abs/2305.10091>



LIST OF PUBLICATIONS

Journals:

Kareem Thajeel, I., Samsudin, K., Jahari Hashim, S., & Hashim, F. (2023). Dynamic feature selection model for adaptive cross site scripting attack detection using developed multi-agent deep Q learning model. *Journal of King Saud University -Computer and Information Sciences*, 35(6), 101490. <https://doi.org/10.1016/j.jksuci.2023.01.012> Q1 Journal- Top 10% - JCR indexed Journal (Published).

Kareem, I. T., Samsudin, K., Jahari, S., & Hashim, F. (2023). Machine and Deep Learning-based XSS Detection Approaches : A Systematic Literature Review. *Journal of King Saud University - Computer and Information Sciences*, 35(7), 101628. <https://doi.org/10.1016/j.jksuci.2023.101628>. Q1 Journal- Top 10% - JCR indexed Journal (Published).

Kareem, I. T., Samsudin, K., Jahari, S., & Hashim, F. (2024). Adaptive Cross-Site Scripting (Xss) Attacks Detection Framework Using Tabu Search-Based Tempo-Spatial Dynamic Feature Selection Approach. (Sumbmitted)

Kareem, I. T., Samsudin, K., Jahari, S., & Hashim, F. (2024). MARL-GDFS: Adaptive Cross-site Scripting Attacks Detection Approach Using Multi-Agent Reinforcement Learning for Group-based Dynamic Feature Selection. (Sumbmitted)

Kareem, I. T., Samsudin, K., Jahari, S., & Hashim, F. (2024). DQN-MAFS: A Deep Q-Network Multi-Agent Feature Selection Model for Adaptive Cross Site Scripting Attack Detection. (Sumbmitted)