DEVELOPMENT AND VALIDATION OF HYBRID ALGORITHMS FOR SOFTWARE DEFECT PREDICTION

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY
by

SONALI CHAWLA
Roll No.2K19/PHDCO/10

Under the Supervision of
PROF. RUCHIKA MALHOTRA
Professor and Head of Department,
Department of Software Engineering
Dr. ANJALI SHARMA
Senior Principal Scientist
Planning Monitoring Evaluation, CSIR-NPL, India

Department of Software Engineering

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi 110042

March, 2026

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Shahbad Daulatpur, Main Bawana Road, Delhi-42

CANDIDATE'S DECLARATION

I, **Sonali Chawla (2K19/PHDCO/10)**, hereby certify that the work which is being presented in the thesis entitled **"Development and Validation of Hybrid algorithms for Software Defect Prediction"** in the partial fulfillment of the requirements for the award of the Degree of Doctor of Philosophy, submitted in the **Department of Software Engineering**, Delhi Technological University, is an authentic record of my own work carried out during the period from **2019** to **2026** under the supervision of **Prof. Ruchika Malhotra** and **Dr. Anjali Sharma**.

The matter presented in the thesis has not been submitted by me for the award of any other degree of this or any other Institute.

Sonali Chawla

This is to certify that the student has incorporated all the corrections suggested by the examiners in the thesis and the statement made by the candidate is correct to the best of our knowledge.

Signature of Supervisors **Signature of External Examiner**

DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)

Shahbad Daulatpur, Main Bawana Road, Delhi-42

CERTIFICATE BY THE SUPERVISOR

Certified that **Sonali Chawla (2K19/PHDCO/10)** has carried out their research work presented in this thesis entitled **"Development and Validation of Hybrid algorithms for Software Defect Prediction"** for the award of **Doctor of Philosophy** from the Department of Software Engineering, Delhi Technological University, Delhi, under our supervision. The thesis embodies results of original work, and studies are carried out by the student herself, and the contents of the thesis do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other Institution.

Supervisor

PROF. RUCHIKA MALHOTRA

Professor and Head of Department

Department of Software Engineering

Delhi Technological University, Delhi-110042

Co-Supervisor

Dr. ANJALI SHARMA

Senior Principal Scientist

Department of Planning Monitoring and Evaluation

CSIR-NPL, Delhi-110012

Date:

Acknowledgment

First and foremost, I bow in gratitude to the Almighty, whose grace, blessings, and guidance have been with me throughout this journey. It is through divine strength and faith that I have been able to persevere and complete this significant milestone in my life.

I would like to express my heartfelt gratitude to my supervisor and HOD (Department of Software Engineering), **Prof. Ruchika Malhotra** for her unwavering support, insightful guidance, and consistent encouragement throughout the course of my research. Her clarity of thought and commitment to academic excellence helped shape the direction and depth of my work. I am especially thankful to her for facilitating access to NVIDIA GPUs and other essential computational resources, which played a crucial role in successfully executing my deep learning and metaheuristic experiments. Her mentorship has strengthened my technical foundation and inspired me to approach research with greater rigor and confidence.

I am deeply grateful to **Dr. Anjali Sharma** for her continuous encouragement, thoughtful feedback, and compassionate mentorship. Her ability to guide with both intellectual depth and personal warmth has supported my growth not only as a researcher but also as an individual. She has been a steady source of motivation during moments of uncertainty, helping me refine my ideas and push beyond my perceived limits. Her mentorship has left a lasting impact on my academic journey, and I will always remain thankful for the opportunity to learn and grow under her guidance.

My heartfelt gratitude goes to my husband, **Abhinav Singhal**, for being my unwavering support system. His love, encouragement, and belief in my dreams have been my strength through the highs and lows. His patience and sacrifices during this demanding phase of life have meant the world to me.

To my precious daughters, **Saanvi Singhal** and **Jhanvi Chawla** your innocent smiles and boundless love brought light to my darkest days. You came into my life

during this journey and gave me a renewed purpose and strength to carry on.

I extend my deepest thanks to my parents **Mr. Gulshan Chawla** and **Mrs. Renu Chawla** for their unconditional love, values, and the countless sacrifices they have made to see me reach this stage. Their prayers, trust, and constant encouragement have always uplifted me. A special thanks to my brother **Sahil Chawla**, whose quiet presence and support have always brought me comfort and motivation.

Lastly, I extend my heartfelt thanks to my entire **family** and **friends**, who have been my constant source of strength throughout this journey. Their unwavering support, encouragement, and countless ways of assisting me - whether through thoughtful conversations, emotional backing, or simply being there - played a crucial role in helping me stay focused and complete this milestone. This thesis is not just a product of academic rigor, but a reflection of the collective love, patience, and belief of those who stood by me at every step.

Sonali Chawla

Abstract

Software defect prediction (SDP) is an important research subject aimed at improving the reliability, maintainability, and overall quality of software systems. The rapid development of software projects raises the need for robust and accurate predictive models. While traditional machine learning (ML) and statistical methods have shown promise for SDP, challenges like high-dimensional data, imbalanced data, inefficient feature selection, and model-tuning limitations persist. To overcome these limitations, this research focuses on the development and validation of hybrid algorithms that leverage the power of both machine learning and metaheuristic optimization techniques to improve predictive performance capabilities for SDP. The research is validated through systematic review, empirical studies, and the development of novel algorithms applicable in real-world software development environments.

The research is systematically structured into phases, addressing distinct components of SDP. The initial phase involves a synthesis of a systematic literature review that seeks to evaluate the latest hybrid algorithms that enhance the predictive performance of SDP models and identify research gaps. The review develops a framework for analyzing the current state-of-the-art with respect to hybrid algorithms on multiple dimensions and highlights the gaps that this thesis will work to address. In subsequent phases, the research develops and validates several novel hybrid algorithms using benchmark datasets from repositories such as NASA, PROMISE, and AEEEM. These later phases include addressing the prime issues of dataset imbalance, designing improved feature selection techniques, implementing hyper-parameter tuning,

and evaluating the proposed hybrid models against established baseline methods to demonstrate their effectiveness in real-world software defect prediction scenarios.

The high-dimensional software datasets greatly influence the efficiency and accuracy of predictive models. Feature selection plays a vital role in simplifying complex datasets while retaining the most significant information. A hybrid SDP model integrating Binary Particle Swarm Optimization (BPSO), Synthetic Minority Oversampling Technique (SMOTE), and Artificial Neural Network (ANN) is proposed to improve software quality. One of the significant contributions of this research is the development of a hybrid defect prediction framework that integrates filter feature selection(Information Gain, Relief F, and Chi-square) and metaheuristic optimization(Opposition-based Whale Optimization Algorithm) for feature selection with attention-based deep learning classifier- Convolutional Neural Networks (1Dimensional- CNN), to achieve higher classification performance. This model is particularly valuable when dealing with large datasets, complex feature interactions, and the need for balancing multiple objectives, such as maximizing classification performance while minimizing the number of features.

Predictive models for SDP often underperform when using default configurations, highlighting the critical need for hyperparameter optimization in maximizing model effectiveness. In this research work, we employed advanced optimization techniques, specifically Grey Wolf Optimization (GWO) and Salp Swarm Optimization(SSO) algorithms, in combination with machine learning and ensemble classifiers to create more effective hybrid models for SDP. These nature-inspired techniques navigate complex parameter spaces to achieve an effective balance between exploration and exploitation in an optimization process. This study highlights that appropriate hyperparameter tuning can yield a significant performance improvement because each predictive model undergoes comprehensive testing for different combinations of parameters before the optimal parameters are reached for each predictive model.

Based on the promising outcomes of the hybrid algorithms developed for defect

prediction, we further investigate their effectiveness by evaluating various hybrid approaches across multiple datasets to ensure the model 's generalizability. The experimental results are favourable for the hybrid models, which outperform traditional ML and statistical defect prediction models. This superiority is evident across key performance metrics, like F1-score, AUC-ROC, Recall, Precision, G-mean, and MCC. Furthermore, rigorous statistical testing confirms the reliability and robustness of these advanced techniques, reinforcing their effectiveness in SDP.

In conclusion, this research significantly progresses the field of SDP by addressing key predictive modelling challenges through the development and validation of sophisticated hybrid techniques. The study strengthens the effectiveness, reliability, and real-world applicability of defect prediction models. This study offers innovative methods for enhancing software quality, which benefits both academia and industry. The insights generated from this research provide a foundation for future advancements in predictive modelling, which will eventually help create software systems that are more dependable, efficient, and free of flaws.

Contents

List of Tables **vii**

List of Figures **xi**

List of Publications **xiii**

Abbreviations **xv**

1 Introduction **1**

1.1 Introduction 1

1.1.1 Software Quality 3

1.1.2 Software Defect 4

1.1.3 Software Defect Management 4

1.1.4 Software Defect Prediction 6

1.1.5 Metaheuristic Algorithms for SDP 10

1.2 Literature Survey 13

1.2.1 Software Metrics 13

1.2.2 Software Defect Prediction 15

1.2.3 Software Defect Prediction using Hybrid algorithms 17

1.3 Objectives of the Thesis 20

1.3.1 Vision 20

1.3.2 Focus 20

1.3.3 Goals 21
1.4 Overview of the Work 23
1.5 Organization of the Thesis 26

2 Research Methodology **29**
2.1 Introduction 29
2.2 Research Process 30
2.3 Defining Research Problem 32
2.4 Literature Survey 33
2.5 Defining Variables 33
2.5.1 Independent Variables 34
2.5.2 Dependent Variables 34
2.6 Data Analysis Methods 35
2.6.1 Support Vector Machine 35
2.6.2 Artificial Neural Network 37
2.6.3 Convolutional Neural Network 39
2.6.4 Ensemble learners 41
2.6.5 Binary Particle Swarm Optimization 43
2.6.6 Grey Wolf Optimization 44
2.6.7 Opposition-based Whale Optimisation (OBWOA) 45
2.6.8 Salp Swarm Optimization 48
2.7 Experimental Design 49
2.7.1 Data collection 50
2.7.2 Dataset Details 51
2.7.3 Data Preprocessing 54
2.7.4 Feature Selection 56
2.7.5 Data Balancing 59
2.7.6 Predictive Model Development and Validation 60

2.7.7 Performance Evaluation Metrics 61
2.7.8 Statistical Analysis 64

3 Systematic Literature Review 69
3.1 Introduction 69
3.2 Review Procedure 71
3.3 Review Protocol 72
3.3.1 Search Strategy 73
3.3.2 Inclusion and Exclusion Criteria 74
3.3.3 Quality Assessment Criteria 75
3.4 Review Results 80
3.4.1 Results specific to RQ1 80
3.4.2 Results specific to RQ2 84
3.4.3 Results specific to RQ3 85
3.4.4 Results specific to RQ4 87
3.4.5 Results specific to RQ5 99
3.4.6 Results specific to RQ6 101
3.4.7 Results specific to RQ7 110
3.4.8 Results specific to RQ8 111
3.4.9 Results specific to RQ9 113
3.5 Discussion 115

4 Software Defect Prediction based on Binary Particle Swarm Optimization and ANN 119
4.1 Introduction 119
4.2 Research Methodology 121
4.2.1 Proposed BPSO-based Feature Selection 122
4.2.2 Datasets and Variables 122
4.2.3 Data Preprocessing 122

4.2.4 Model Development and Validation 123
4.2.5 Parameter Settings 124
4.2.6 Performance Measures 124
4.2.7 Statistical Evaluation 125
4.2.8 Baseline Techniques 125
4.3 Results and Analysis 126
4.3.1 Results specific to RQ1 126
4.3.2 Results specific to RQ2 127
4.3.3 Results specific to RQ3 130
4.4 Discussion 132

5 Software Defect Prediction based on Multi-Filter Wrapper Feature Selection and Deep Neural Network with Attention mechanism 135

5.1 Introduction 135
5.2 Research Methodology 138
5.2.1 Feature Selection based on Multi-Filter Wrapper Technique 139
5.2.2 Defect Prediction based on Attention-based CNN 141
5.3 Experimental Setup 144
5.3.1 Dataset collection and Preprocessing 144
5.3.2 Parameter Settings 145
5.3.3 Performance Metrics 145
5.3.4 Statistical Analysis 147
5.4 Result Analysis 147
5.4.1 Result Analysis based on RQ1 148
5.4.2 Result Analysis based on RQ2 151
5.4.3 Result Analysis based on RQ3 154
5.4.4 Result Analysis based on RQ4 160
5.5 Discussion 168

6 Hybrid Model for SDP using parameter tuning of ML classifier 171

6.1 Introduction . . . 171

6.2 Research Methodology . . . 174

6.2.1 Dataset Collection and Variables . . . 174

6.2.2 Data Pre-processing . . . 174

6.2.3 Model Development and validation . . . 174

6.2.4 Performance Metrics . . . 176

6.2.5 Statistical Evaluation . . . 176

6.3 Results and Analysis . . . 176

6.3.1 Result Analysis based on RQ1 . . . 176

6.3.2 Result Analysis based on RQ2 . . . 178

6.3.3 Result Analysis based on RQ3 . . . 178

6.4 Discussion . . . 180

7 Hybrid Model for SDP using parameter tuning of Ensemble classifiers 183

7.1 Introduction . . . 183

7.2 Research Methodology . . . 186

7.2.1 Dataset Collection and Variables . . . 186

7.2.2 Data Preprocessing . . . 186

7.2.3 Model Development and Validation . . . 186

7.2.4 Hyperparameter Settings . . . 188

7.2.5 Performance Metrics . . . 188

7.2.6 Statistical Evaluation . . . 189

7.3 Results Analysis . . . 190

7.3.1 Results analysis based on RQ1 . . . 190

7.3.2 Results analysis based on RQ2 . . . 191

7.3.3 Results analysis based on RQ3 . . . 192

7.4 Discussion . . . 193

8 Conclusion **197**

8.1 Summary of the Research Work 197

8.2 Application of the Work 201

8.3 Future Work 203

Bibliography **213**

Supervisor's Biography **244**

Author's Biography **246**

List of Tables

2.1 Comparison of Dataset Repositories Used in SDP Research 51
2.2 Statistical Summary of NASA Datasets 52
2.3 Statistical Summary of AEEEM Datasets 53
2.4 Statistical Summary of PROMISE Datasets 54

3.1 Quality Assessment of Reviewed Studies 77
3.2 Primary Studies with Study Number, Quality Score, Reference, and Citation 78
3.3 Publication Distribution by Type and Impact Factor 87
3.4 Filter feature selection techniques 92
3.5 Wrapper feature selection techniques 93
3.6 Parameter optimization techniques 94
3.7 Performance measures used by primary studies 96
3.8 Description of validation methods 98
3.9 Fitness functions employed by search-based techniques in hybrid-based models 100
3.10 Descriptive statistics of performance measures for hybrid-based SDP models 103
3.11 Wilcoxon test results for recall values 108
3.12 Wilcoxon test results for F-measure values 108
3.13 Wilcoxon test results for accuracy values 109

3.14 Statistical tests used by primary studies 111
3.15 Strengths and weaknesses of hybridized techniques for SDP 112
3.16 Threats to validity reported by primary studies 113

4.1 Parameters for the BPSO algorithm 125
4.2 Parameters for the ANN algorithm 125
4.3 Comparison of results - with and without feature selection 127
4.4 Comparison of ANN, BPSO-ANN, and BPSO-SMOTE-ANN models in terms of AUC, Recall, Precision, and G-mean 128
4.5 Average AUC scores of proposed hybrid classifier and other techniques 129
4.6 Average G-mean scores of proposed hybrid classifier and other techniques 130
4.7 Friedman Test results based on AUC and G-mean 130
4.8 Wilcoxon Test results based on AUC and G-mean 131

5.1 Parameter settings for OBWOA 146
5.2 Parameter settings for 1D-CNN 146
5.3 Average selected features obtained by MFWFS 149
5.4 Comparative results of 1D-CNN with and without MFWFS based on F-measure 150
5.5 Comparative results of 1D-CNN with and without MFWFS based on AUC 150
5.6 Comparative results of 1D-CNN with and without MFWFS based on G-mean 150
5.7 Comparative results of 1D-CNN with and without MFWFS based on MCC 150
5.8 Wilcoxon test result based on average F-measure, AUC, G-mean, and MCC 151

5.9 Comparative results of MFWFS-1D-CNN with and without attention based on F-measure 153
5.10 Comparative results of MFWFS-1D-CNN with and without attention based on AUC 153
5.11 Comparative results of MFWFS-1D-CNN with and without attention based on G-mean 153
5.12 Comparative results of MFWFS-1D-CNN with and without attention based on MCC 153
5.13 Wilcoxon test result based on F-measure, AUC, G-mean, and MCC . 154
5.14 Performance of different feature selection methods on 1D-CNN . . 154
5.15 Performance comparison of IG + CS + RF + OBWOA with CNN and BERT 161
5.16 Abbreviations of the algorithms used for comparison with the proposed model 162
5.17 Results of the proposed approach against eighteen other algorithms in respect of AUC 165
5.18 Average performance values of the proposed approach against other models 167

6.1 F-measure and AUC scores 177
6.2 Optimized parameters of SVM 178
6.3 Results of the Friedman test based on F-measure and AUC 180

7.1 Hyperparameter settings for ensemble learning models 188
7.2 Average Performance Metrics Across All Datasets 191
7.3 Statistical Significance Test Results (Wilcoxon Signed-Rank Test) . 193

A.1 Algorithms used in primary studies 206
A.2 List of top 15 highly cited journals/conferences 211

List of Figures

1.1 Software Defect Management Process 6

1.2 Steps in Predictive Modelling 9

2.1 Research Process 31

3.1 Classification of Search-based and ML techniques 82

3.2 Distribution of studies by search-based and ML techniques used . . 83

3.3 Classification of Primary studies by year and publication 85

3.4 Distribution of primary studies according to their type 86

3.5 Distribution of studies according to the type of datasets 89

3.6 Percentage of studies according to software metrics 90

3.7 Distribution of primary studies based on performance metrics 97

3.8 Distribution of studies based on validation methods used 99

3.9 Dataset-wise recall outliers for hybridized techniques 104

3.10 Dataset-wise F-measure outliers for hybridized techniques 104

3.11 Dataset-wise accuracy outliers for hybridized techniques 105

3.12 Distribution of studies according to statistical analysis 110

4.1 Proposed methodology 123

4.2 Algorithm for BPSO-SMOTE-ANN 124

5.1 Proposed methodology 139

5.2 Architecture of attention-based 1D-CNN 144

5.3 Box plot for F-measure, AUC, G-mean, and MCC values for different feature selection methodologies 159

6.1 Proposed methodology 175

6.2 Comparative results based on F-measure 179

6.3 Comparative results based on AUC 179

7.1 Proposed methodology 187

List of Publications

Papers Accepted/Published in International Journals

1. Ruchika Malhotra, Sonali Chawla, Anjali Sharma, "Software defect prediction using hybrid techniques: a systematic literature review", *Soft Computing*, 2023 (Impact factor: 3.1).

2. Ruchika Malhotra, Sonali Chawla, Anjali Sharma, "Software Defect Prediction based on Multi-Filter Wrapper Feature Selection and Deep Neural Network with Attention mechanism", *Neural Computing and Applications*, 2025 (Impact factor:4.5).

Papers Accepted/Published in International Conferences

3. Ruchika Malhotra, Sonali Chawla and Anjali Sharma, "An Artificial Neural Network Model based on Binary Particle Swarm Optimization for enhancing the efficiency of Software Defect Prediction, *In Proceedings of the 2023 6th International Conference on Software Engineering and Information Management (ICSIM '23)*, pp. 92-100, Association for Computing Machinery, New York, NY, USA.

4. Ruchika Malhotra, Sonali Chawla and Anjali Sharma, "Evaluating the effec-

tiveness of metaheuristic techniques for tuning SVM parameters in Software Defect Prediction", *Proceedings of the International Conference on Intelligent Computing and Communication Techniques (ICICCT 2024)* , Delhi, India.

Papers Communicated in International Journals

5. Ruchika Malhotra, Sonali Chawla, Anjali Sharma, "Enhancing Software Defect Prediction Through Meta-heuristic Hyperparameter Optimization of Ensemble Learning Models ", *International Journal of System Assurance Engineering and Management*.

Abbreviations

ML	**Machine Learning**
SMOTE	**Synthetic Minority Oversampling Technique**
KNN	**K-Nearest Neighbors**
RF	**Random Forest**
GB	**Gradient Boosting**
AdaBoost	**Adaptive Boosting**
XGB	**Extreme Gradient Boosting**
CNN	**Convolutional Neural Network**
ANN	**Artificial Neural Network**
SVM	**Support Vector Machine**
PSO	**Particle Swarm Optimization**
GA	**Genetic Algorithm**
OBWOA	**Opposition Based Whale Optimization Algorithm**
GWO	**Grey Wolf Optimization**
SSO	**Salp Swarm Optimization**
MFWFS	**Multi Filter Wrapper Feature Selection**
IG	**Information Gain**

CS	**Chi-Square**
RF	**Relief-F**
ROC-AUC	**Receiver Operating Characteristic - Area Under the Curve**
MCC	**Matthews Correlation Coefficient**
RBF	**Radial Basis Function**
AUC	**Area Under the Curve**
DL	**Deep Learning**
RQ	**Research Question**
LOC	**Lines of Code**
NOA	**Number of Attributes**
NOM	**Number of Methods**
NOC	**Number of Children**
DIT	**Depth of Inheritance Tree**
CBO	**Coupling Between Object Classes**
RFC	**Response for a Class**
LCOM	**Lack of Cohesion of Methods**
OO	**Object-Oriented**
SQA	**Software Quality Assurance**
SDP	**Software Defect Prediction**

Chapter 1

Introduction

1.1 Introduction

Software has become a critical component in various domains, including business, healthcare, medicine, education, and entertainment. As the complexity and size of software systems grow, so does the importance of ensuring their quality and reliability. The presence of defects in software is a major challenge that can cause failure of large engineering projects and lead to significant financial losses. The early identification of software defects can reduce the cost and time of maintenance by allocating the resources judiciously, prioritizing testing efforts to the areas of the system with a higher probability of defects and ensuring high-quality software [1].

Software defect prediction (SDP) is a process of creating a predictive model that helps in identifying the defect-prone software modules before the testing phase begins [2]. Classification algorithms play a crucial role in SDP. The predictive models categorize the software modules into predefined classes, such as defective or non-defective. Researchers have examined the application of several statistical and ML techniques for this purpose. By leveraging the historical defect data from

previous software projects, an efficient predictive model predicts the defective classes in new software modules. However, building accurate and robust predictive models remains challenging due to the complexity of software systems and the variability in development processes.

A complementary approach to SDP is search-based software engineering, which uses metaheuristic techniques that explore large solution spaces to provide optimal or near-optimal solutions to complex problems [3]. Search-based algorithms are robust and capable of handling noisy data very effectively, which is why they are appropriate for developing predictive models. They can search for efficient solutions by modelling performance metrics as *"fitness functions"*[4]. Hybrid models are formed by combining ML and search-based techniques into a single approach. The strengths of both algorithms are combined while mitigating individual limitations. Hybrid models are able to attain higher accuracy, robustness, and generalization capabilities as opposed to individual models when multiple approaches are combined.

This thesis investigates the use of several hybrid algorithms for developing software defect prediction models. The focus of this research is to contribute to the development of new techniques, effective predictive models and their evaluation for determining the defect-prone classes to obtain high-quality and reliable software. This chapter consists of the basic concepts involved and the motivation for the work. In section 1.1.1, we describe the concept of software quality. In Section 1.1.2 and Section 1.1.3, we define the basic terminology of software defects and the process of software defect management. In Section 1.1.4, we discuss what SDP is, the predictive modelling process and the issues faced in predictive modelling. Section 1.1.5 discusses the metaheuristic and hybrid algorithms used for SDP. Section 1.2 conducts a literature survey. The remainder of the chapter discusses the objectives of the thesis (Section 1.3), an overview of the work(Section 1.4) and its organisation (Section 1.5).

1.1.1 Software Quality

According to the Institute of Electrical and Electronics Engineers (IEEE), software quality is defined as [5]:

- *"The degree to which a system, component, or process meets specific requirements."*
- *"The degree to which a system, component, or process meets customer or user needs or expectations."*

High-quality software must meet all its desired attributes - functional that focuses on what software does, such as user authentication, search functionality, data validation, etc. and non-functional aspects that focus on how software performs, like efficiency, scalability, and reliability. Functional attributes ensure that the software performs its intended tasks correctly and meets user requirements. Non-functional attributes ensure the support of effective delivery of functional attributes. By addressing these attributes, organizations can enhance the quality and performance of their software products, leading to increased customer satisfaction.

Software quality encompasses both *"quality of design"* and *"quality of conformance"*[6]. The quality of design is related to the adequateness of software design. An appropriate software design is required to create software with minimal defects and a high level of customer satisfaction. The degree to which software adheres to the developed design is related to the quality of conformance.

Software quality can be determined in terms of various software quality attributes like functionality, maintainability, usability, reliability, testability, and adaptability. Apart from these, software defect proneness is one quality attribute in the domain of software engineering, and this thesis limits itself to this attribute. It refers to the probability or likelihood of a software module or class containing defects or errors

after the software is released. A software class/module can be found defective/non-defective in the subsequent release of the product. It is an important concept in software quality assurance and defect management, as it helps identify parts of the software that are more likely to cause issues and therefore require more rigorous testing and review. Thus, it is important to remove those defects as soon as possible to ensure customer satisfaction.

1.1.2 Software Defect

A software defect is a fault or a bug in the program that creates a deficiency in the software product, which produces incorrect and unexpected results [7]. It is the error in the software product that is a deviation from the requirement specification or end-users' expectations. In software development, a lot of defects would emerge during the development process. These defects not only decrease the quality of the software but also increase the cost due to delays in the development schedule. Effective testing is required to capture software defects to maintain the product's quality, efficiency, and reliability. Early defect detection prevents defect migration from the requirement phase to design and from the design phase into the implementation phase.

Approximately 20% of all software defects take 80% of the time and effort required to analyze, isolate and fix software defects [8]. Therefore, it is critical to assess and monitor the quality of the software product thoroughly.

1.1.3 Software Defect Management

Software defect management is a systematic process for identifying, tracking, prioritizing, resolving, and preventing defects in software products. Effective defect management ensures that defects are handled efficiently and helps improve the overall quality of the software. The primary objective of software defect management is to

improve software quality by detecting and fixing defects in the early stages of the software development lifecycle. In developing high-quality software, software defect prediction plays a vital role. Identification of defects in the initial stages of SDLC is complicated, hence efficient methods should be applied to eliminate them. There are 5 main stages of handling defects as shown in Figure 1.1 [9] :

1. Defect Identification - Defect identification is the process of discovering the occurrence of defects in software. It involves various techniques and methods to uncover errors, inconsistencies, or deviations from expected behaviour. Key methods include testing, code inspections, and manual reviews.

2. Defect Categorization - It is a critical step that involves classifying defects into various categories according to different criteria like severity, type, cost of fixing, and the feature in which defects are discovered. A defect goes through various stages during its lifecycle-

    - Active
    - Test
    - Verified
    - Closed

3. Defect Analysis - Defect analysis involves identifying the root cause of defects and implementing measures to prevent their recurrence in subsequent projects. This step improves the software development process and enhances overall software quality.

4. Defect Prediction - It involves the early identification of potential defect-prone modules in a software development lifecycle. This approach helps to allocate testing and quality assurance resources to high-risk areas, thereby preventing defects before they occur.

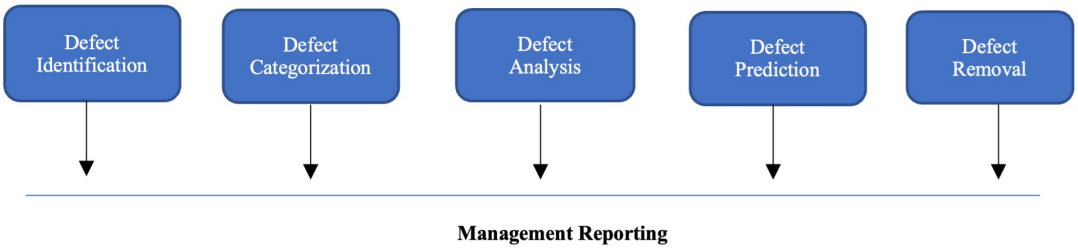

Figure 1.1: Software Defect Management Process

5. Defect Removal - It is a systematic and structured approach to identifying and fixing software system defects according to priority and severity. This ensures that the software meets its required quality standards and performs as expected.

1.1.4 Software Defect Prediction

Among the various stages of software defect management outlined in the above section, this research focuses specifically on the defect prediction phase, which is crucial for proactively detecting potential faults early in the development cycle, ultimately helping to minimize cost, effort, and the risk of software failures. SDP is a key area in software engineering that involves forecasting the likelihood of defects in software components before they manifest in the system. The prediction of software defects is based on the knowledge that if a project developed in an environment leads to faults, then any module developed in a similar environment with similar project features will end up being defective [10]. It is a process of developing a predictive model that is capable of identifying software modules in a project as defective or non-defective based on the underlying properties of the prior software projects. An SDP model is either a classification model that categorizes modules/classes into defective/non-defective or a regression model that predicts the number of defects in a module or class. Software organizations can use these SDP models during the early stages of the software development lifecycle to identify faulty modules to save time, increase

customer satisfaction, and ensuring a good quality product in the end. The developers and testers can shift their focus on the high-risk faulty components of the project which reduces the effort and time to review the code and helps to choose the right design and provide reliable, low-cost, and maintainable software [11]. This work encompasses classification techniques where SDP is considered as a binary problem.

1.1.4.1 Predictive modelling for SDP

Predictive modelling is a statistical and computational technique used to forecast future outcomes or events. It involves a process that analyses the historical data and the current data to generate models that predict target variables (or future outcome) based on one or more different input variables or attributes. Using predictive modelling, the companies are able to better understand customer behaviour, financial, economic and market risks. For instance, banks and financial institutions can use predictive models to detect fraudulent activities. The model analyzes transaction patterns and flags unusual behaviour that could indicate fraud. This can protect customers from fraudulent transactions and thus minimize financial losses.

In the realm of SDP, predictive modelling plays a crucial role in identifying potential defects in software systems before they are released into production. By analyzing historical data related to software development, such as code metrics, process metrics, and defect logs, predictive models help in forecasting which components or modules are likely to contain defects. This proactive approach allows developers and testers to focus their efforts on the most vulnerable parts of the software. Predictive models contribute to continuous quality improvement by identifying patterns and trends in defect occurrence, leading to more effective quality assurance strategies.

Figure 1.2 describes the process followed in SDP through predictive modelling.

1. **Identify Objectives:** This involves understanding the goals of SDP, like defect

classification - classifying the software modules or classes into defective or non-defective.

2. **Define Research Objectives:** This involves formulating the research questions that the predictive model will address, like identifying the key performance measures - accuracy, AUC, precision, recall, F1 score, etc. - and comparing the model with baselines.

3. **Preparation of Datasets:** The historical data needs to be collected from previous projects in the form of software metrics like lines of code, cyclomatic complexity, code churn, etc and defect logs. The data is organized in a structured format suitable for analysis.

4. **Data Preprocessing:** The data is prepared for modelling by handling null, missing values and outliers. The data is normalized to ensure consistent values across all the features are present. Feature selection is performed to select the most relevant features using different techniques like filter, wrapper and hybrid methods. The data is split into training, validation, and testing to ensure the model is evaluated properly.

5. **Model selection:** The choice of the model depends on the type of data, attributes and prediction problem (classification in the case of a binary SDP problem), like machine learning (ML) algorithms, deep learning (DL) algorithms, or hybrid algorithms.

6. **Model Training:** The training dataset is fed into the model, which involves fitting the model to the data and optimizing the parameters to achieve optimum performance.

7. **Model Validation:** The model is evaluated on the testing dataset, measuring

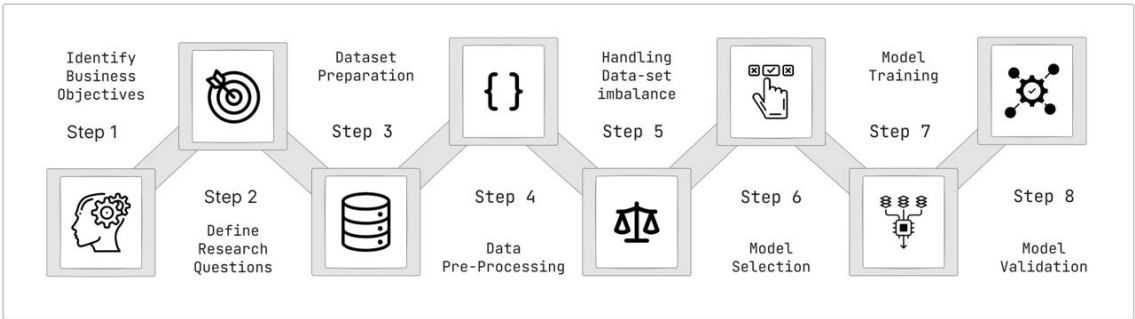

Figure 1.2: Steps in Predictive Modelling

different performance measures like AUC, F1-score, Recall, Precision and accuracy. Statistical tests are conducted to validate the results of the model.

The predictive models for SDP are developed with various techniques such as statistical, machine learning(ML) and Hybrid algorithms that learn from historical software data and predict defects for the future versions of the product.

1.1.4.2 Challenges in Predictive Modelling for SDP

While preparing the datasets for predictive modelling in SDP, there are many software metrics inspected by the researchers based on the software development process or code complexity [12]. However, not all the features are equally relevant for prediction. Redundant or irrelevant features can increase model complexity, leading to overfitting and degraded model performance. Choosing the right subset of features is critical for improving the accuracy and efficiency of the model since projects have different requirements and different development procedures. Traditional predictive models can show limited classification performance as their performance is heavily dependent on model parameters. Tuning techniques like grid search or random search are often time-consuming and inefficient, especially when dealing with high-dimensional hyperparameter spaces. They may miss the optimal configuration or require significant computational resources. Another significant issue in predictive

modelling is the imbalanced dataset problem. This problem is more prevalent in the domain of software engineering due to the high imbalance ratio of defective and non-defective instances in software defect datasets. Most software modules are non-defective, which leads to models being biased towards predicting the majority class (non-defective) while underperforming in detecting the minority class (defective).

The challenges in predictive modelling â including feature selection complexities, inefficient parameter tuning, and class imbalance â will require advanced optimization methods capable of navigating complex solution spaces. Hybrid algorithms offer promising solutions to address these issues because they can effectively search large solution spaces and locate near-optimal solutions without getting stuck in local optima. In this research work, we have devoted our effort to improve defect prediction modelling performance through hybrid algorithms (a combination of metaheuristic and machine/deep learning techniques) to address the feature selection and parameter tuning problem. The software defect datasets that are highly imbalanced are also taken care of using different data resampling techniques in the preprocessing stage to develop effective prediction models. This study also encompasses statistical tests to strengthen the results that have been reportedly less evident in previous studies.

1.1.5 Metaheuristic Algorithms for SDP

Metaheuristic procedures are search-based techniques that can find an optimized solution from a broad search space consisting of possible solutions. When these algorithms are applied to software engineering problems, they give rise to search-based software engineering and are treated as search-based optimization problems or simply search problems. Search-based software engineering has been applied throughout the software lifecycle, beginning from requirements and project planning to maintenance and re-engineering. A search problem is one in which optimal or

near-optimal solutions are explored in a candidate solution search space, driven by a fitness function that differentiates between better and worse solutions [13]. Harman and Jones [14] identified two primary components for the application of search-based optimization to software engineering problems. i) representation of the problem ii) fitness function definition. Representation of a problem is necessary, as without it, one cannot do engineering if a problem is not represented properly. Also, software engineering problems contain a variety of software metrics that can serve initially as a good candidate for fitness functions. Once these two tasks are fulfilled, this gives rise to the applicability of many search-based optimization techniques based on the nature of software engineering problems. This generality and applicability form one advantage of search-based engineering. Scalability is achieved through the parallel execution of fitness functions.

Metaheuristic techniques are increasingly applied in SDP due to their flexibility and effectiveness in optimizing complex, multi-dimensional problems. In the context of SDP, metaheuristics can enhance predictive modelling in several ways. Metaheuristic techniques help in identifying the most relevant subset of features, improving model accuracy and reducing computational costs. By evaluating different combinations of features, these algorithms optimize the subset that maximizes predictive performance for a given model. These algorithms are effective in hyperparameter tuning. Intelligently exploring the hyperparameter space, these algorithms maintain a balance between exploration and exploitation to find near-optimal settings that improve the model's predictive performance. Various metaheuristic algorithms can automate the search for optimal neural network architectures. These techniques search for the best combination of layers, neurons, activation functions, and other hyperparameters, resulting in neural architectures tailored specifically for SDP tasks.

1.1.5.1 Hybrid Algorithms

Search-based techniques are categorized into different categories, such as Local search, Swarm intelligence, Evolutionary, and Hybrid [15]. There are a variety of metaheuristic algorithms belonging to these categories that are applied in the area of software engineering, such as genetic algorithms, simulated annealing, ant colony optimization, particle swarm optimization, hill climbing, tabu search, etc. When these search-based techniques are used in combination with machine learning algorithms, it is known as a hybrid algorithm or a hybridized search-based technique. Mostly, swarm intelligence and evolutionary techniques are used in hybridized techniques. Hybrid techniques typically integrate the strengths of both search-based and ML techniques into a single approach. The hybrid techniques may use a search-based technique for either hyperparameter tuning , feature selection, or to decide a structure of another technique to achieve optimal results [15]. Hybrid techniques provide a global optimum solution without getting trapped in local minima [16]. They are well-suited for projects with different properties and characteristics [13]. They undergo a simplified automated search process with fewer parameters required. Their convergence speed is fast, leading to quality optimal solutions [17][18]. Harman and Clark [4] explained how the relationship between metrics and fitness function can act as a guiding force in the search for optimal or suboptimal solutions in solving a software engineering problem. Several researchers have applied these hybrid algorithms in the area of defect prediction. Many studies show the popularity of hybrid algorithms in defect prediction as they combine the benefits of both machine learning and search-based techniques.

1.2 Literature Survey

SDP has become essential for delivering quality software products by identifying potential issues before they can significantly impact functionality or user experience. Researchers have spent years refining methodologies and techniques to enhance both defect prediction accuracy and efficiency in this critical area. A comprehensive review of the existing body of literature should be conducted to identify the current research gaps, finding opportunities for innovation and propelling towards advancement in the field.

This section discusses various software metrics that have been used in the literature to predict software defects. It also surveys diverse models that researchers have proposed to predict software defects, highlighting the methodologies that were employed.

1.2.1 Software Metrics

Software metrics play a crucial role in SDP as they serve as the primary features or attributes used to build predictive models. These metrics provide measurable characteristics of software that can be analyzed to identify patterns associated with software defects. In the literature, a wide range of software metrics has been employed, which focus on the quality aspects of the product, process and project [19].

1.2.1.1 Product Metrics

Product metrics are the measurable attributes of a software product, focusing on the properties of the software such as its size, complexity, and quality, assuming complex software components are more prone to contain bugs. Product metrics are typically derived from the source code, design documents, or other deliverables produced during

the software development lifecycle. Throughout the history of software engineering, various code metrics have been used for SDP.

Size Metrics: Size-oriented metrics measure the quantity of the code based on the lines of code computation. They are directly obtained from the source code, hence also known as code metrics [20]. Many authors have used lines of code [21] as the metric for developing SDP models [22] [23] [24]. While they provide valuable insights, they are insufficient on their own for accurately predicting software defects.

Halstead Metrics: Halstead's metrics [25] evaluate the software complexity based on interpreting the source code as a sequence of tokens and classifying each token to be an operator or an operand. This metrics suite consist of following metrics: (i) Number of unique operators (n_1) (ii) Number of unique operands (n_2) (iii) Program Vocabulary: the sum of number of unique operators and operands (iv) Program length (N) is the sum of the total number of operators and operands in the program (v) Program volume is given as: $V = N \cdot \log_2 \eta$ (vi) Program Level is given as: $L = V * /V$ (vii) Program difficulty: D = $(n_1/2) * (N_2/n_2)$. The studies that have used Halstead metrics for developing SDP models are [26] [27] [28].

McCabe's Metrics: McCabe's metrics [29] are founded upon graph theory. This metric suite consists of the following metrics: (i) Cyclomatic Complexity ($v(G)$), (ii) Essential Complexity ($ev(G)$), (iii) Design Complexity ($iv(G)$), and (iv) Cyclomatic Density. With these metrics, many researchers build various SDP models [30] [31] [32] [33].

Chidamber and Kemerer (CK) [34] proposed a metrics suite that consists of six metrics about various aspects of OO software. These metrics are Weighted Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number Of Children (NOC), Coupling Between Object (CBO), Response For a Class (RFC), and Lack of Cohesion in Methods (LCOM). This metric suite has been validated in various studies predicting software defects [35] [36] [37].

1.2.1.2 Process Metrics

Process metrics have emerged as powerful predictors of software defects by capturing the dynamic aspects of software development. These metrics measure how software is developed and maintained over time, including code churn (lines added/modified/deleted), developer activity patterns, and change history [38]. Studies have shown that process metrics often outperform traditional product metrics in defect prediction [39] [40]. Key process metrics include change frequency, change size, number of developers involved in modifications, and code review coverage [1]. Recent research has also highlighted the value of repository mining metrics like commit patterns and developer expertise [41]. Datasets like AEEEM and PROMISE include rich process metrics, enabling researchers to model defect-prone areas by correlating changes in code over time with defect occurrence. Integration of process metrics with traditional product metrics has significantly improved defect prediction models [42].

1.2.2 Software Defect Prediction

The literature has demonstrated the connection between software metrics and defect prediction. Many models have been put out in the literature to predict software defects.

McCabe [43] introduced cyclomatic complexity as a measure of program complexity, demonstrating that modules with complexity values exceeding 10 were more likely to contain defects, and modules with complexity above 20 were considered high-risk. Halstead's [25] software metrics, showing that programs with higher volume and difficulty metrics tended to have more defects due to increased cognitive load on developers. Fenton and Ohlsson [43] provided empirical validation across multiple releases of a large telecommunications system, showing that while LOC had a basic correlation with defects, the combination of LOC with McCabe's and Halstead's metrics provided better defect prediction capability.

The introduction of object-oriented metrics by Chidamber and Kemerer [34] revolutionized defect prediction for OO systems - their landmark paper introduced the CK metrics suite, demonstrated strong correlations between these metrics and fault-proneness. Building on this, Basili et al. [44] validated the CK metrics suite empirically, showing that classes with high coupling and low cohesion were more likely to contain defects. Subramanyam and Krishnan [45] specifically examined size metrics alongside OO metrics, demonstrating that size alone was insufficient for accurate defect prediction, but when combined with inheritance and coupling metrics, prediction accuracy improved significantly. More recent work by Zhou and Leung [46] focused on inheritance-based metrics, revealing that deeper inheritance hierarchies correlated with higher defect probability, though this relationship plateaued beyond certain depths. D'Ambros et al. [1] conducted a comprehensive evaluation comparing traditional metrics (McCabe, Halstead) against OO metrics, finding that while both sets had predictive power, combining them with process metrics like change history yielded the best results. They particularly highlighted that Response for Class (RFC) and Coupling Between Objects (CBO) were among the strongest individual predictors. Hall et al. [47] performed a systematic review of 208 fault prediction studies, revealing that models using combinations of metrics generally outperform those using single metrics, with size, complexity, and process metrics being the most reliable predictors when used together.

SDP models have employed various traditional machine learning techniques over the years, with significant research validating their effectiveness. Support Vector Machines (SVM) [48] demonstrated superior prediction performance compared to eight other machine learning models (Naïve Bayes, Logistic Regression, RBF Networks, Multilayer Perceptron, Random Forests, Decision Trees, Bayesian Belief Networks, and K-Nearest Neighbor) for SDP. The study used four NASA datasets (CM1, KC1, KC3, and PC1) to validate their findings. SVM consistently showed

better accuracy, precision, recall, and F-measure across all datasets. Lessmann et al. [49] performed a comprehensive empirical evaluation by examining 22 classifiers across 10 publicly available NASA datasets. They discovered that the majority of classification models achieved comparable performance levels based on AUC score, with no statistically significant differences among the top 17 classifiers. Malhotra and Singh [50] employed Artificial neural networks, Logitboost, Adaboost, Naive Bayes, Bagging, KStar, logistic regression, and random forest to predict defects on the open-source dataset Arc. It was determined that Logitboost performed the best, with an ROC-AUC of 0.806. Jayanthi and Florence [51] proposed a hybrid approach utilizing PCA as a feature reduction technique and ANN for classification. They conducted their experiments on publicly available NASA datasets (CM1, KC1, KC2, PC1), and for comparative analysis with various state-of-the-art techniques in terms of area under the curve (AUC). The results show the importance of feature reduction and preprocessing and the proposed approach show improved AUC scores.

1.2.3 Software Defect Prediction using Hybrid algorithms

In addition to statistical and machine learning techniques, hybrid algorithms represent a newly evaluated category of algorithms for SDP. These algorithms combine machine learning and search-based strategies to leverage their complementary strengths. Harman [52] argued that evolutionary algorithms are especially well-suited for software defect prediction due to their capability to manage the complex, multiobjective nature of software quality assessment. Although numerous studies have explored machine-learning techniques for defect prediction extensively, the exploration of hybrid algorithms remains largely uncharted. Researchers are investigating hybrid algorithms, with their current applications primarily focused on feature selection [53] [54] [55], parameter tuning [56] [57], or model development [58] [59].

Jin & Jin's [18] study proposed a hybrid approach combining Artificial Neural Networks (ANN) with Quantum Particle Swarm Optimization (QPSO) for software fault prediction. The QPSO was used for dimensionality reduction, and the ANN for prediction. The approach was experimentally validated on four NASA datasets, which show improved AUC values over traditional ANN, PSO-ANN model and other state-of-the-art techniques.

Malhotra [60] assessed 16 hybrid algorithms for predicting defective classes using 17 datasets based on OO metrics obtained from the Promise Repository. The results were statistically validated and confirmed the effectiveness of hybridized techniques for developing SDP models.

Malhotra and Khan [61] proposed a recent study in which they implemented a two-phase mutation Grey Wolf Optimizer for feature selection using five machine learning classifiers: SVM, Random Forest, Gradient Boosting, AdaBoost, and KNN. They employed 27 open-source software datasets, which were balanced using the Synthetic Minority Oversampling Technique (SMOTE). The results were statistically evaluated based on AUC scores, and the proposed approach outperformed existing algorithms, achieving the best performance.

Zhu et al. [62] introduced a novel feature selection method, EMWS, to select optimal software metrics for defect prediction and a hybrid deep learning model called WSHCKE that integrates Convolutional Neural Network (CNN) and Kernel Extreme Learning Machine (KELM) for defect classification. The experiments were conducted on 20 open-source datasets in which EMWS achieved 31% improvement in F1-score and 65%improvement in MCC compared to 11 baseline feature selection methods. WSHCKE outperformed nine baseline classifiers, showing 25% higher F1-score and 35% higher MCC.

Turabeih et al. [63] proposed a novel approach for SDP that uses three feature selection algorithms (Binary Genetic Algorithm, Binary Particle Swarm Optimization,

and Binary Ant Colony Optimization) to identify the most relevant software metrics for fault prediction. These are combined with an L-RNN classifier. The approach was tested on 19 real software projects from the PROMISE repository. The results reveal that cross-validation with feature selection shows excellent classification performance with an average AUC of 0.8358 across datasets.

Li et al. [64] proposed a hybrid model that combines a modified bat algorithm called Changing Range Bat Algorithm (CRBA) to optimize Support Vector Machine (SVM) parameters for software defect prediction. The CRBA-SVM model achieved the best overall performance compared to other traditional methods and hybrid variants of SVM, with a maximum accuracy value of of 93% and a maximum F-measure of 96.3%.

Kassaymeh et al. [65] research study finds the optimal parameters of BPNN using the Salp Swarm Algorithm to improve the defect prediction accuracy. The experiment was conducted on 22 different datasets and evaluated using a cross-validation technique. This resulted in an exceptional performance compared to traditional BPNN and thirteen other state-of-the-art techniques.

De Carvalho et al. [58] introduced a novel approach for SDP using multiobjective Particle Swarm Optimization (MOPSO) that generates rules for predicting faulty software modules. It used sensitivity and specificity as optimization objectives. The approach was tested on NASA datasets and evaluated using AUC and other metrics like accuracy, precision, and recall. MOPSO gave competitive results compared with Neural and Bayesian Networks and achieved the highest values of AUC and accuracy in almost all the datasets.

1.3 Objectives of the Thesis

1.3.1 Vision

Improving software quality by developing hybridized SDP models in open source software systems.

1.3.2 Focus

The focus of the work has been to evaluate and improve software quality models, considering defect as the quality attribute. In this regard, effective SDP models are developed from open-source systems. This thesis endeavours to conduct research in predictive modelling using various software metrics and hybrid techniques. Keeping in mind the challenges of imbalanced data while developing the prediction models, the current study carefully handles the imbalanced datasets to develop effective SDP models. The cross-validation technique and statistical tests have been used while developing the models to ensure that the results are unbiased. Thus, this study explicitly addresses the following perspectives

1. To analyze the significance of hybrid algorithms available in the literature by examining their essential components.
2. To investigate the effectiveness of hybrid algorithms and compare their performance with other ML and statistical algorithms.
3. To devise a new hybridized technique by improvising the existing technique in the literature.
4. To explore the application of CNN and deep neural networks in defect prediction.

5. To investigate whether one hybrid technique outperforms others in terms of performance measures with the help of statistical tests.

1.3.3 Goals

The summarization of the goals investigated in this work is provided below:

1. Perform a systematic literature review on SDP using Hybrid Algorithms
    - Study of existing research publications that would help to understand the process and procedure of developing SDP models. The systematic literature review will help to understand the effectiveness of hybrid techniques in the area of defect prediction.
    - An extensive study of existing literature to understand the significance of proposed hybridized techniques that are related to defect prediction modelling.
    - Study of various types of hybrid algorithms used in literature for improving the performance of defect prediction models, with their strengths and weaknesses.
    - A review of hybrid algorithms for feature selection and parameter optimization used in building quality prediction models and evaluating their performance.
2. Perform data collection for empirical validation of the defect prediction models
    - To collect diverse software defect datasets from established repositories.
    - To analyse dataset characteristics like the size of the datasets, the programming language the data is using, class imbalance ratios, feature distributions, and defective module percentages.

3. Explore, compare and evaluate hybrid algorithms and evaluate their effectiveness for developing prediction models

    - To explore hybrid algorithms that combine the machine learning algorithms and search-based algorithms into a single approach and assess their capability for defect prediction models.
    - To evaluate the performance of each technique to determine its performance efficacy
    - To compare the performance of different hybrid algorithms with other machine learning and statistical algorithms by performing statistical analysis to identify the best-performing algorithms for developing a prediction model.

4. Explore and evaluate deep learning and CNN-based techniques for defect prediction

    - To explore the application of deep learning neural networks and a convolutional neural network model in the area of SDP.
    - To evaluate the deep learning and CNN models for defect prediction to judge the predictive ability of these techniques and assess their performance.

5. Develop and implement a hybridized algorithm for effective improvement in the performance of defect prediction models

    - To develop a new hybridized algorithm for building an SDP model.
    - To evaluate the overall performance of the proposed approach.
    - To compare the new proposed hybridized algorithm with other existing algorithms in terms of performance and efficiency.

1.4 Overview of the Work

SDP is a crucial process in software development that helps identify potential defects in software modules or code sections before they become critical problems. Anticipating defects early in the development process can help developers strategically allocate testing resources, reduce maintenance costs, and improve overall software reliability. Therefore, organizations can avoid expensive post-release problems, optimize project schedules, refine testing strategies, and enhance their quality assurance practices over time. This proactive approach to defect prediction is essential for delivering reliable, high-quality software that aligns with user expectations while minimizing long-term maintenance costs and operational risks.

This section outlines the work conducted in this research study. A thorough systematic literature review was performed, adhering to the guidelines proposed by Kitchenham [66], to gain a clear understanding of prior research in the domain of SDP. The primary goal of this review was to analyze and comprehend the role of hybrid techniques in SDP studies from multiple perspectives:

- Commonly used search-based and ML techniques that form hybrid techniques for SDP model development.
- Datasets, metrics, feature/parameter optimization, and validation methods used in the design of hybridized techniques for SDP.
- various fitness functions employed in hybrid techniques for SDP,
- performance measures used for the evaluation of hybrid techniques for SDP,
- overall and comparative performance of various hybrid techniques with statistical and ML techniques,

- different statistical tests used to validate the results of hybrid techniques for SDP,
- strengths and weaknesses of these techniques,
- potential threats to validity as reported by the studies.

After formulating a search string, the most relevant and credible digital libraries were selected to retrieve pertinent research papers related to the study. The data extracted from these studies was analyzed systematically to address the research questions effectively.

In SDP, managing high-dimensional datasets poses a significant challenge, as it can greatly influence the efficiency and accuracy of predictive models. Feature selection play a vital role in simplifying complex datasets while retaining the most significant information. We proposed a hybrid SDP model integrating BPSO, SMOTE, and ANN to improve software quality. The model effectively selects features, handles dataset imbalance, and predicts defect-prone software modules with improved AUC and G-mean values across five different open-source datasets. Statistical validation reinforces the reliability of the results, showcasing their potential to lower maintenance costs and improve the overall quality of software development processes.

Predictive modelling in SDP faces several cchallenges,like dataset imbalance and limited performance of traditional machine learning techniques, along with the presence of redundant and irrelevant metrics. We proposed a novel hybrid model based on a multi-filter-wrapper feature selection technique and 1D-CNN with an attention mechanism. By integrating different filter feature selection techniques like information gain, chi-square, Relief-F methods, and the wrapper technique Opposition-based WOA, the approach effectively selects optimal features from 17 publically available software datasets. The model's performance was comprehensively assessed using four performance measures: AUC, G-mean, MCC, and F-measure. The study' s

results were benchmarked against existing state-of-the-art algorithms and statistically validated, confirming the model's robustness and effectiveness in SDP.

Parameter optimization plays a vital role in enhancing the performance and reliability of SDP models by fine-tuning algorithm parameters to achieve optimal predictive accuracy and generalizability. This chapter highlights the importance of advanced parameter optimization techniques, which address challenges such as imbalanced datasets, overfitting, and convergence issues. Effective parameter optimization not only improves the performance of predictive models but also ensures their robustness across diverse datasets and scenarios, leading to more accurate identification of defect-prone modules and contributing to the overall quality of software development. One study uses GWO and SSO to tune SVM parameters that lead to improved predictive performance over classical SVM and traditional parameter tuning methods. The next study advances the first one by using the same metaheuristic aalgorithms,GWO and SSSO,to optimize the ensemble learning models RandomForest, Gradient Boost, AAdaboost, andXGBoost. Both the above studies were conducted using five AEEEM software datasets that demonstrated the significance of parameter optimization in SDP that ensures the model's robustness, which ultimately leads to more accurate identification of defect-prone modules and contributes to the overall quality of software development.

1.5 Organization of the Thesis

This section outlines the organization of the thesis. **Chapter 1** establishes the foundational concepts of the work and presents the research motivation. **Chapter 2** details the research methodology required to accomplish the research objectives. **Chapter 3** presents a comprehensive systematic review of existing literature and identifies the prevailing research gaps. **Chapter 4** presents the development of a hybrid SDP model that focuses on feature selection. **Chapter 5** proposes a new hybrid SDP model using a unique combination of feature selection methods coupled with a deep neural network with an attention mechanism. **Chapter 6** develops hybrid SDP models utilizing the two hyperparameter tuning techniques on the SVM ML classifier. **Chapter 7** extends the previous research by developing hybrid SDP models using hyperparameter tuning techniques based on ensemble learning classifiers. In **Chapter 8**, the conclusions of the thesis are presented. A brief description of each chapter is given below. **Chapter 1**: This chapter establishes the foundational framework for the research investigation into SDP within the software engineering domain. It systematically addresses the challenges associated with predictive modelling in software quality, focusing on efficient feature selection, parameter tuning, and handling imbalanced data. It presents how hybrid algorithms emerge as potential solutions to these multifaceted problems. The chapter also presents the research objectives, carefully delineating the scope of the study and the meaningful contributions of the research. Furthermore, it describes the detailed steps involved in developing SDP.

Chapter 2: This chapter presents a thorough overview of the research methodology used to achieve the given objectives. It gives a brief explanation of dependent and independent variables, followed by classification techniques and hybrid algorithms used in the study. The experimental design covers the data collection, preprocessing

steps, imbalance handling of the data, and validation methods used for developing the SDP model. The performance evaluation metrics and statistical methods used to validate the study are also presented.

Chapter 3: This chapter presents a systematic literature review that investigates 72 primary studies (2000-2021) on hybrid techniques for SDP. It offers a comprehensive analysis of existing research in software quality , emphasizing the role of hybrid algorithms in defect prediction. Research questions are formulated, and the studies are summarized to offer insights into various aspects of hybrid techniques, including datasets, feature selection methods, parameter tuning techniques, fitness functions, validation techniques, evaluation metrics, predictive performance of different hybrid techniques, and statistical tests used in these studies. It examines the strengths and weaknesses of existing studies, identifies gaps and lays the foundation for potential improvements in this field.

Chapter 4: This chapter proposes a hybrid SDP model integrating Binary Particle Swarm Optimization (BPSO), Synthetic Minority Oversampling Technique (SMOTE), and Artificial Neural Network (ANN) to improve software quality. The model effectively selects features, handles dataset imbalance, and predicts defect-prone software modules with higher predictive performance across five different datasets. Statistical validation confirms the significance of the results in reducing maintenance costs and enhancing software development quality.

Chapter 5: This chapter proposes a novel hybrid model based on a multi-filter-wrapper feature selection technique and a 1D-convolutional neural network with an attention mechanism. By combining Information Gain, Chi-square, Relief-F methods, and Opposition based Whale Optimization Algorithm, the approach effectively selects optimal features from 17 software datasets. The model outperforms existing methods across four performance metrics, demonstrating enhanced classification performance, and statistical analysis proves the validity of the results.

Chapter 6: This chapter optimizes SDP using meta-heuristic optimization algorithms Grey Wolf Optimization (GWO) and Salp Swarm Optimization (SSO) to tune Support Vector Machine (SVM) parameters. The hybrid GWO-SVM and SSO-SVM models outperformed classic SVM and traditional parameter tuning methods like Grid Search and Random Search. Statistical validation proved that comprehensive parameter optimization significantly improves predictive performance, reducing overfitting and enhancing computational efficiency.

Chapter 7: This chapter explores GWO and SSO for tuning ensemble learning models - Random Forest, Gradient Boosting, AdaBoost, and XGBoost in SDP on five AEEEM datasets. The study demonstrates significant performance improvements concerning three performance evaluation metrics, highlighting the importance of hybrid algorithms in hyper-parameter optimisation.

Chapter 8: The concluding chapter summarizes the thesis's key findings and contributions, emphasising their significance for both academic research and industry practices. It highlights the advancements achieved in software quality prediction and their potential impact on the field. Additionally, the chapter outlines future research opportunities, offering recommendations for further investigation and development to address existing challenges and explore new directions.

Chapter 2

Research Methodology

2.1 Introduction

A well-defined and structured research methodology is fundamental for ensuring the credibility, reproducibility, and validity of empirical results. In the context of this study, which focuses on enhancing SDP through hybrid algorithms that include advanced machine learning models and metaheuristic optimization techniques, the research methodology serves as a comprehensive framework that guides the investigation from problem identification to experimental validation.

This chapter outlines the sequential components of the methodology. It begins by presenting the overall research process in Section 2.2 and defining the core research problem in Section 2.3. In Section 2.4, a thorough literature survey is conducted to establish the theoretical foundation and identify existing gaps in the field. The chapter further elaborates on the definition and selection of variablesâboth independent and dependentâin Section 2.5, which are central to model development and evaluation. Subsequently, the methodology describes the analytical techniques employed in Section 2.6. The chapter also details the experimental design in Section 2.7, covering

data collection in Section 2.7.1, dataset details in Section 2.7.2. data preprocessing in Section 2.7.3, feature selection in Section , data balancing in Section 2.7.5, and predictive model development and validation procedures in Section 2.7.6, performance metrics are examined in Section 2.7.7, and statistical analysis methods are covered in Section 2.7.8.

By systematically addressing each methodological component, this chapter ensures a clear and rigorous pathway for evaluating the effectiveness of the proposed hybrid models and optimization strategies. The approach adopted herein is intended to maintain scientific rigour while facilitating a meaningful comparison of predictive techniques within the domain of software quality assurance.

2.2 Research Process

The research process comprises a carefully organized series of steps aimed at systematically investigating and addressing the research problem. This structured methodology provides a coherent framework, ensuring that each phase of the study - from the initial identification of the issue to the experimental analysis and final conclusions - progresses logically and efficiently. Figure 2.1 offers a visual overview of the research process, mapping out the interconnected stages that are woven throughout the chapters of this thesis to promote consistency and transparency.

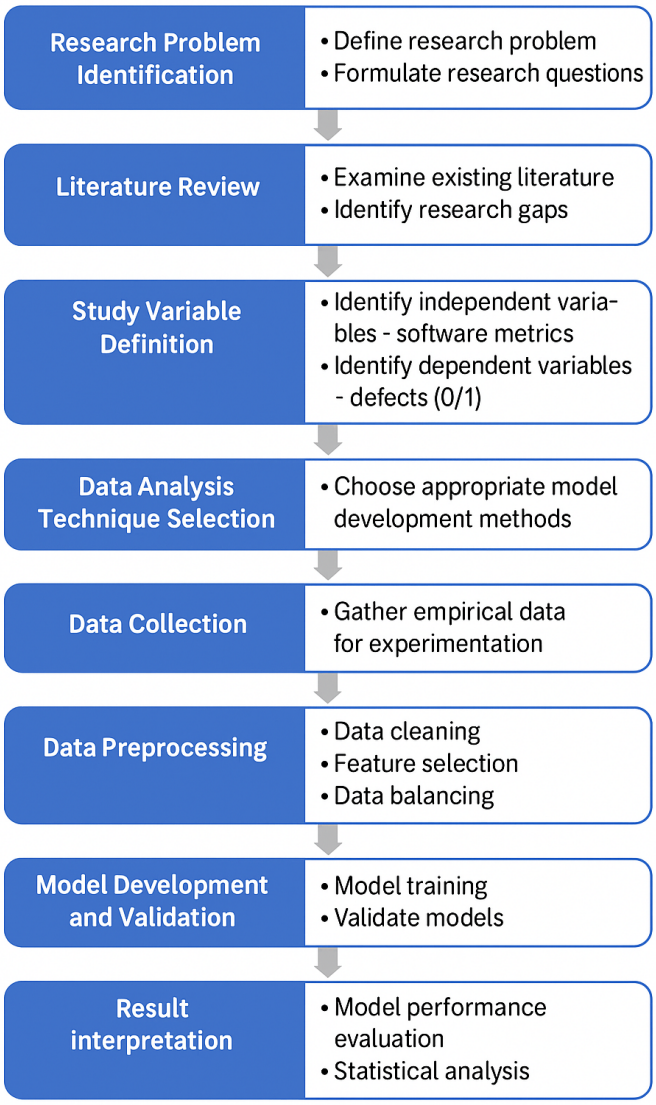

Figure 2.1: Research Process

Each phase plays an integral role in supporting a rigorous and in-depth investigation. The subsequent sections present a detailed account of each stage, explaining the specific activities conducted, how they are interrelated, and their contributions to achieving research objectives. Following this systematic approach ensures that the study remains focused and well-structured while also preserving scientific rigour and

methodological validity. This, in turn, strengthens the overall credibility and reliability of the empirical findings.

2.3 Defining Research Problem

The first stage of the research process is focused on defining the research problem. At this stage, the primary topic under investigation has been precisely determined and articulated through thoughtfully constructed research questions (RQs). These questions provide a focused framework that directs the course of the research. By establishing clear RQs, the research lays the groundwork for designing experiments that systematically address the core issues.

A set of research questions has been developed in this thesis to enhance understanding of the main research problem and aid in the achievement of the study's objectives. These questions play a critical role in shaping the overall methodology, including the selection of data, analytical techniques, and experimental procedures, thereby ensuring that the findings are both meaningful and robust. These questions are:

1. What is the current state of literature on hybrid algorithms, and what research gaps exist regarding their application in Software Defect Prediction(SDP)?

2. Which hybrid algorithms can researchers use to develop efficient SDP models from imbalanced data?

3. How do hybrid feature selection techniques influence the performance of SDP models?

4. How does parameter tuning in hybrid algorithms influence the robustness and generalization performance of SDP models?

5. How do hybrid models perform when compared to traditional single-technique ML/DL approaches or other hybrid algorithms in SDP?

2.4 Literature Survey

It is crucial to conduct a literature survey of existing studies to gain a comprehensive understanding of the research problem. Through this review, we can assess how extensively hybrid models have already been investigated in the existing body of work to improve defect prediction performance. In recent years, numerous models [18, 60, 62, 64, 65, 67, 68] have been proposed to predict software defects, many of which utilize hybrid techniques (optimization techniques in conjunction with ML/DL). These models often establish relationships between internal software attributes, captured through software metrics, and the presence or absence of defects. Both commercial and open-source project datasets have served as the foundation for developing and validating these models. The predictive capabilities of hybrid SDP models are particularly valuable during the early stages of software development, as they enable practitioners to identify defect-prone modules in advance. Early detection of such modules allows project managers to allocate testing and maintenance resources more effectively, thus reducing the likelihood of post-release defects. As emphasized in the existing literature, the development of robust hybrid SDP models is critical for improving software reliability and overall quality.

2.5 Defining Variables

For any empirical study, it is essential to clearly identify two types of research variables: independent and dependent variables. Independent variables are the factors or predictors that influence the outcome of the study. They are used to forecast the

value of the dependent variable and should ideally be uncorrelated with each other to minimize bias in the model [69]. In this research, software metrics serve as the independent variables that describe the measurable characteristics of the software. In this thesis, we investigate the association of these independent variables derived from well-established software metric suites with software defects. The dependent variables are the outcome or target variable that the research aims to predict, often referred to as the response variable, discussed in subsection 2.5.1. In the context of this thesis, the dependent variable is the presence of software defects, which is discussed in detail in subsection 2.5.2.

2.5.1 Independent Variables

The independent variables that are utilized in SDP models are typically comprised of a wide variety of software metrics. These metrics form the basis of the evaluation and prediction of the existence of defects in software systems. Section 1.2.1 of this thesis offers a thorough discussion of these metrics, including their definitions, classifications, and roles in defect prediction. This section explores each statistic in detail, providing comprehensive information on how it has been estimated and measured.

2.5.2 Dependent Variables

The dependent variable, also known as the target variable or response variable, is an essential part of the modelling process in SDP as it dictates the outcome the model seeks to predict. This variable typically reflects whether a software module, file, or unit of code contains defects, and it serves as the foundation for assessing the predictive model's performance[70]. A widely adopted method is to use binary classification for defect status. Here, the dependent variable divides components into

two categories: defective (1) and non-defective (0). This binary approach dominates traditional defect prediction research, where the goal is to flag high-risk components using historical data and software metrics (e.g., code complexity, cohesion). Its simplicity aids interpretability and implementation, particularly when prioritizing testing efforts on defect-prone code sections.

2.6 Data Analysis Methods

This thesis incorporates hybrid techniques for the development of prediction models that combine various ML/DL/ensemble learning models with search-based techniques. This leads to unified algorithms that improve classification performance by combining the strengths of multiple techniques within a single, integrated framework. Machine learning methods are data-driven techniques that learn patterns from historical data to model the association between independent and dependent variables, enabling the prediction of future outcomes. Deep learning methods effectively capture complex, non-linear relationships in large-scale data, further improving model accuracy. Ensemble learning techniques enhance predictive robustness by aggregating the outputs of multiple base classifiers to form a more reliable final prediction. Search-based techniques are especially well-suited for solving optimization problems, thereby enhancing the predictive performance of the models.

2.6.1 Support Vector Machine

Support Vector Machines (SVMs) are powerful supervised learning algorithms widely used for classification and regression tasks in machine learning and data mining. SVMs operate by identifying an optimal hyperplane that separates data points of different classes with the maximum possible margin, thereby improving the model's

ability to generalize to unseen data [71]. The data points that are closest to this hyperplane are known as support vectors, and they play a critical role in defining the decision boundary. SVMs are particularly effective in high-dimensional spaces and maintain robustness against overfitting, especially in cases where a clear margin of separation exists.

In many real-world scenarios, the data is not linearly separable in its original feature space. To address this, SVMs employ kernel functions, which implicitly map the input data into a higher-dimensional feature space where a linear separation becomes feasible. This transformation is done without explicitly computing the coordinates in the higher-dimensional space, a technique known as the kernel trick.

Common kernel functions include:

- Linear Kernel: Suitable for linearly separable data, or when the number of features is high relative to the sample size.
- Polynomial Kernel: Allows the algorithm to fit more complex, non-linear boundaries by capturing polynomial relationships between features; particularly useful when feature interactions are important.
- Radial Basis Function (RBF) or Gaussian Kernel: The most widely used kernel for non-linear problems; it measures the similarity between two points based on their distance.
- Sigmoid Kernel: Behaves like activation functions in neural networks, but is less commonly used due to practical constraints.

The choice of kernel can significantly affect the accuracy and performance of SVM models, and the best kernel often depends on the specific characteristics of the dataset being analyzed.

The effectiveness of SVMs is further determined by several critical parameters:

- Regularization Parameter (C): The parameter C controls the trade-off between maximizing the margin and minimizing classification error. A smaller value of C encourages a wider margin but may tolerate more misclassifications, while a larger C seeks to classify all training examples correctly, potentially at the expense of overfitting.

- Kernel-specific Parameters: Each kernel function introduces its own set of parameters. For instance, the RBF kernel includes the gamma (γ) parameter, which defines the influence of a single training example. A low γ value results in smoother, more generalized decision boundaries, whereas a high γ value yields more complex, localized boundaries[72]. Similarly, the polynomial kernel's degree parameter determines the flexibility of the decision boundary.

Optimal performance of SVMs is typically achieved through systematic parameter tuning, often employing grid search and cross-validation techniques to identify the most suitable combination of C, kernel type and kernel-specific parameters[73].

2.6.2 Artificial Neural Network

An Artificial Neural Network (ANN) is a computational model inspired by the structure and functioning of the human brain, consisting of interconnected processing elements called neurons. ANNs are particularly well-suited for complex pattern recognition and prediction tasks, making them a popular choice for SDP [74]. The basic structure comprises an input layer, one or more hidden layers, and an output layer, with each layer containing multiple neurons that perform specific computational operations. The typical process involves training the ANN on historical project data, where input features represent software metrics (e.g., LOC, cyclomatic complexity, cohesion, and coupling metrics), and the output indicates whether a module is defective or not [75].

Once trained, the network can predict the defect-proneness of new or unseen modules, thus supporting early detection and targeted quality assurance efforts.

The mathematical foundation of neural networks involves the computation of weighted sums of inputs, followed by the application of activation functions to introduce non-linearity into the model [73]. For a given neuron j in layer l, the output is computed as:

$$z_j^{(l)} = \sum_{i=1}^{n} w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)} \tag{2.1}$$

$$a_j^{(l)} = \phi\left(z_j^{(l)}\right) \tag{2.2}$$

Where:

- $a_j^{(l)}$ is the activation (output) of neuron j in layer l, after applying the activation function ϕ.
- $w_{ij}^{(l)}$ is the weight associated with the connection from neuron i in the previous layer $l-1$ to neuron j in layer l
- $a_i^{(l-1)}$ is the activation of neuron i in layer $l-1$,
- $b_j^{(l)}$ is the bias term for neuron j in layer l,
- $\phi(\cdot)$ is the activation function (e.g., sigmoid, ReLU, tanh)[76].

The performance of an ANN is highly dependent on several key parameters and architectural choices.

- No. Of layers and No. Of neurons per layer: This parameter determines the network's ability to model complex relationships between software metrics and defect proneness.

- Activation function: This parameter introduces the non-linear transformation of input data within each layer and ultimately impacts convergence behaviour and prediction accuracy.
- Learning rate: This parameter controls the step size during weight updates.
- Epochs: This parameter defines the number of times the entire training dataset is passed through the network.
- Batch size: This parameter determines the number of training samples processed before updating network weights.

2.6.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a specialized class of deep learning models designed to process data with a grid-like topology, such as images, but also increasingly applied to structured data in domains like SDP [77]. CNNs are distinguished by their unique architecture, which enables automatic extraction of hierarchical features from raw input, reducing the need for manual feature engineering. These models are suitable for image classification, object detection, image recognition, etc., in computer vision tasks [62]. A typical CNN is composed of three primary types of layers: convolutional layers, pooling layers, and fully connected layers.

- Convolutional Layer: This layer performs the core operation of convolution, which applies a filter (kernel) to an input matrix of pixels (representing an image) to produce a feature map. The sliding function applied to the matrix is called a ”kernel” or ”filter,” and both terms can be used interchangeably. The output from the convolution operation can be mathematically expressed as

$$Z_{i,j}^{(k)} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X_{i+m,j+n} \cdot W_{m,n}^{(k)} + b^{(k)} \tag{2.3}$$

- $Z_{i,j}^{(k)}$: Output value at position (i, j) in the k-th feature map
- X: Input (software metrics)
- $W^{(k)}$: Kernel (filter) weights for the k-th filter
- $b^{(k)}$: Bias term for the k-th feature map
- $M \times N$: Size of the filter

This operation slides the filter over the input to extract local patterns.

- Pooling Layer: Pooling layers reduce the spatial dimensions of the feature maps, typically by taking the maximum or average value within a local window (max pooling or average pooling). The maximum pooling operation selects the maximum value from a defined window over the activation map.
- Fully Connected Layers: The high-level abstracted features from the last convolutional or pooling layer are flattened and passed through one or more dense layers. If x is the input vector from the previous flattened layer

$$z = Wx + b \tag{2.4}$$

$$\hat{y} = \sigma(z) \tag{2.5}$$

- W: Weight matrix
- b: Bias vector
- σ: Activation function (commonly ReLU for hidden layers, sigmoid for the final layer)
- $\hat{y}$: Predicted output (probability of defect)

2.6.4 Ensemble learners

Ensemble learning is a machine learning approach that integrates several base models (weak learners) to generate a resilient, high-performance predictive model. As opposed to single-model approaches, ensemble learners mitigate overfitting, reduce variance, and improve generalization by utilizing the diversity of individual learners [78, 79]. The four prominent ensemble techniques studied in this thesis are Random Forest, Gradient Boosting, XGBoost, and AdaBoost.

Random Forest is a parallel ensemble method based on bagging (bootstrap aggregating). It constructs multiple decision trees during training, each trained on a random subset of the training data (with replacement) and a random subset of features. The final prediction is typically made by majority voting in classification tasks or averaging in regression tasks [80, 81]. Random Forest is valued for its robustness, ease of use, and ability to handle both classification and regression problems. The key hyperparameters optimized in this thesis include the number of trees, maximum depth, and minimum samples per split. The decision function for Random Forest is given by equation (2.6):

$$f(x) = \frac{1}{n} \sum_{i=1}^{n} h_i(x) \tag{2.6}$$

where $h_i(x)$ represents individual decision trees.

Gradient boosting is a sequential ensemble method in which each subsequent model is developed to rectify the faults of its predecessors. Gradient boosting generates a strong predictive model from several weak learners by emphasizing the mistakes of previous learners, often decision trees [82]. Although this approach is well-known for its excellent accuracy and adaptability, improper regularization may make it susceptible to overfitting. The hyperparameters optimized in this thesis include the

learning rate, number of estimators, and tree depth. The model update function is given in equation (2.7):

$$F_m(x) = F_{m-1}(x) + \nu \cdot \gamma_m h_m(x) \tag{2.7}$$

Where ν is the learning rate, γ_m is the optimal step size, and $h_m(x)$ is the base learner.

AdaBoost is an ensemble method that combines weak learners iteratively, assigning higher weights to misclassified instances in successive iterations [83]. This adaptive approach helps in improving model performance, particularly in scenarios with class imbalances. The primary hyperparameters considered for optimization in this thesis include the number of estimators and the learning rate. The equation for model prediction is given below:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{2.8}$$

where $h_t(x)$ is the weak classifier's prediction, and α_t represents the weight of each weak classifier.

XGBoost (Extreme Gradient Boosting) is a sophisticated version of gradient boosting that integrates enhanced regularisation (L1 & L2) and optimisation methods to augment performance and scalability. XGBoost is widely recognized for its speed, efficiency, and superior predictive power, making it a popular choice in many machine learning competitions and practical applications[84]. This algorithm constructs a novel tree by considering both left and right splits simultaneously, leading to more balanced trees and improved performance.The hyperparameters optimized in this thesis include the learning rate, maximum depth, the number of boosting rounds, and subsamples. The objective function combines training loss and regularization, as given in equation (2.9):

$$\text{Obj}(\theta) = L(\theta) + \Omega(\theta) \tag{2.9}$$

where $L(\theta)$ represents the training loss and $\Omega(\theta)$ is the regularization term.

2.6.5 Binary Particle Swarm Optimization

This thesis uses the binary variant of PSO developed by Kennedy and Eberhart [85, 86] for reducing dimensionality. In BPSO, the particles move in a discrete binary search space. Two major characteristics are considered necessary when dealing with an optimisation problem - representation of solutions and fitness function [87].

The candidate solutions are represented as a one-dimensional vector such as $\{x_1, x_2, x_3, x_4, \ldots, x_n\}$ where n denotes the total number of features. It consists of 0's and 1's, with '1' indicating that a feature is selected and '0' indicating that a feature is not selected. A particle p has a position and a velocity defined by a position vector $Z_p = (Z_{p1}, Z_{p2}, \ldots, Z_{pn})$ and velocity vector $V_p = (V_{p1}, V_{p2}, \ldots, V_{pn})$. A fitness function is applied to each particle, which directs the search for the best solution.

After the swarm's fitness evaluation, the new velocity of all particles is illustrated by a function based on the swarm's global best position and each particle's personal best position using Equation 2.10. All particle positions are then updated using this new velocity as defined by Equation 2.11.

$$V_p(t_1) = \omega \cdot V_p(t_0) + k_1 \cdot r_1 \cdot (Z_{p,\text{best}}(t_0) - Z_p(t_0)) + k_2 \cdot r_2 \cdot (Z_{\text{global}}(t_0) - Z_p(t_0)) \tag{2.10}$$

$$Z_p(t_1) = Z_p(t_0) + Z_p(t_1) \tag{2.11}$$

Here, each particle's initial velocity lies in the range [0, 1]. t_0 and t_1 denote the

current and next iteration, $Z_{p,\text{best}}$ represents the personal best position for particle p and Z_{global} represents the global best position for the entire swarm. ω is the inertia weight, k_1, k_2 are positive constants, and r_1, r_2 are random numbers in the range [0,1]. Since the particles can take only binary values, Z_p, $Z_{p,\text{best}}$, and Z_{global} can each have a value of either 0 or 1 in Equation 2.11. The sigmoid function is used to update the particle's location by converting the particle's velocity value to a probability in the range [0, 1].

The fitness function is selected to minimize the set of features while maximizing the classification performance (AUC) of a model [88]. It is described using Equation 2.12, where α is a user-defined hyperparameter $\in [0, 1]$ that balances both objectives. ρ is the AUC measure, $|S|$ is the feature subset length, and $|N|$ is the total number of features. Based on previous literature, $\alpha = 0.99$ is used in this thesis [88, 89].

$$\text{Fitness} = \alpha \cdot (1 - \rho) + (1 - \alpha) \cdot \left(\frac{|S|}{|N|}\right) \tag{2.12}$$

2.6.6 Grey Wolf Optimization

The social hierarchy and hunting techniques of grey wolves served as an inspiration for the development of a metaheuristic algorithm known as Grey Wolf Optimizer (GWO) in 2014 [90]. GWO mimics the cooperative hunting behaviour observed in wolf packs to efficiently search for optimal solutions to optimization problems. Mathematically, the wolf behaviour is modelled by randomly initializing the population which consists of grey wolves, with each wolf representing a potential solution. There are four hierarchical levels in the population - alpha (α), beta (β), delta (δ), and omega (ω) wolves. α, β, and δ wolves represent the leaders, whereas ω wolves are the ordinary members. The hunting strategy includes three phases: tracking, chasing, and attacking

prey.

During each iteration, the positions of α, β, and δ wolves are continuously updated using equations (2.13), (2.14), (2.15) to mimic the hunting behaviour of the wolves.

$$D_\alpha = |C_1 \cdot Z_\alpha - Z|, \quad D_\beta = |C_2 \cdot Z_\beta - Z|, \quad D_\delta = |C_3 \cdot Z_\delta - Z| \tag{2.13}$$

$$Z_1 = Z_\alpha - P \cdot D_\alpha, \quad Z_2 = Z_\beta - Q \cdot D_\beta, \quad Z_3 = Z_\delta - R \cdot D_\delta \tag{2.14}$$

$$Z_{t+1} = \frac{Z_1 + Z_2 + Z_3}{3} \tag{2.15}$$

Where D_α, D_β, and D_δ represent the distances of a particular wolf (or solution candidate) from the α, β, and δ wolves, respectively. C_1, C_2, C_3 and P, Q, R are the coefficient vectors calculated using the equations:

$$C_1, C_2, C_3 = 2a \cdot r_1 - a \tag{2.16}$$

$$P, Q, R = 2 \cdot r_2 \tag{2.17}$$

r_1 and r_2 are random vectors in the range $[0, 1]$, and $a = 2 - \frac{2t}{T_{\max}}$ is a linearly decreasing parameter from 2 to 0 over the course of iterations.

2.6.7 Opposition-based Whale Optimisation (OBWOA)

- Whale Optimization Algorithm - It is a stochastic population-based metaheuristic algorithm recently invented in 2016 [91]. This algorithm mimics the intelligent search strategy adopted by humpback whales, known as bubble-net feeding, for searching their food. They create bubbles in a spiral motion around their prey, i.e., small fish, and then slowly swim toward the surface. There are two stages in a WOA algorithm: the first involves *encircling the prey* and using

a *spiral-bubble net attacking mechanism* (the exploitation stage), and the second involves *randomly searching for prey* (the exploration stage) [92].

- Exploitation phase
  - Shrinking Encircling Mechanism: The humpback whales swim in a spiral-like pattern as they approach their prey in a shrinking encircling mechanism. They update their positions and move towards the best solution according to the current optimal solution, using the equations below:

$$\vec{D} = \left| \vec{C} \cdot \vec{Z}^{t}_{\text{best}} - \vec{Z}^{t} \right| \tag{2.18}$$

$$\vec{Z}^{t+1} = \vec{Z}^{t}_{\text{best}} - \vec{A} \cdot \vec{D} \tag{2.19}$$

    where $\vec{Z}^{t}_{\text{best}}$ denotes the position of the best solution thus far. The coefficients $\vec{A}$ and $\vec{C}$ can be computed using the equations:

$$\vec{A} = 2 \cdot \vec{a} \cdot \vec{r} - \vec{a} \tag{2.20}$$

$$\vec{C} = 2 \cdot \vec{r} \tag{2.21}$$

    where $\vec{a} = 2 - \frac{2t}{M}$ is linearly decreased from 2 to 0, M refers to the maximum number of iterations, and $\vec{r}$ is a random vector in $[0, 1]$.
  - Spiral path updating position: The bubble-net feeding mechanism calculates the distance between the current and the best solution thus far using the spiral equation (2.22), as illustrated below.

$$\vec{Z}^{t+1} = \vec{D}' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{Z}^{t}_{\text{best}} \tag{2.22}$$

where b defines the constant that dictates the shape of the logarithmic spiral function, and l represents a random value in $[-1, 1]$.

According to equation (2.23), humpback whales have a 50% chance of randomly selecting either a spiral-shaped path model or a shrinking encircling mechanism to approach their prey.

$$\vec{Z}^t = \begin{cases} \vec{Z}^t_{\text{best}} - \vec{A} \cdot \vec{D}, & \text{if } p < 0.5 \\ \vec{D}' \cdot e^{bl} \cdot \cos(2\pi l) + \vec{Z}^t_{\text{best}}, & \text{if } p \geq 0.5 \end{cases}, \quad p \in [0, 1] \tag{2.23}$$

– Exploration Phase

This phase is controlled by the variable $\vec{A}$ that lies between $[-1, 1]$. If $|\vec{A}| \geq 1$, the position of each whale is modified according to the position of a randomly chosen whale, and if $|\vec{A}| < 1$, the position of each whale is modified according to the current optimal solution. This global search capability of WOA is shown in the equations below:

$$\vec{D} = \left| \vec{C} \cdot \vec{Z}^t_{\text{rand}} - \vec{Z}_t \right| \tag{2.24}$$

$$\vec{Z}^{t+1} = \vec{Z}^t_{\text{rand}} - \vec{A} \cdot \vec{D} \tag{2.25}$$

where $\vec{Z}_{\text{rand}}$ is the position of a whale selected randomly from the current population.

• Opposition-based learning

This technology was proposed by Hamid R. Tizhoosh [93], [94], and it has been proven to greatly enhance the performance of optimization algorithms like PSO [95], DE [96], GWO [97], ACO [98], by evaluating the current and

opposite solutions simultaneously. It increases the exploitation of the search space and ultimately helps to reach a globally optimal solution more quickly. The following equations are used to compute the opposite values.

Opposite Number: Assume $z \in [a, b]$ is a real number, where a and b represent the lower and upper limits. The opposite number $\tilde{z}$ is defined as:

$$\tilde{z} = a + b - z \tag{2.26}$$

Opposite Point: Assume $Z(z_1, z_2, z_3, \ldots, z_n)$ is a point where $z_i \in [a_i, b_i]$ for $i = 1, 2, 3, \ldots, n$. The opposite point $\tilde{Z}(\tilde{z}_1, \tilde{z}_2, \ldots, \tilde{z}_n)$ is defined as:

$$\tilde{z}_i = a_i + b_i - z_i \tag{2.27}$$

Opposition-Based Optimization: Let $Z(z_1, z_2, z_3, \ldots, z_n)$ be a candidate solution in the search space, with objective function $f(Z)$. Let $\tilde{Z}(\tilde{z}_1, \tilde{z}_2, \ldots, \tilde{z}_n)$ be its opposite, with fitness function $f(\tilde{Z})$. Then:

$$\text{If } f(\tilde{Z}) > f(Z), \text{ then } Z \leftarrow \tilde{Z}; \text{ otherwise, retain } Z \tag{2.28}$$

2.6.8 Salp Swarm Optimization

The bio-inspired optimization algorithm draws inspiration from the collective foraging behaviour of marine organisms called salps. In 2017, Mirjalili et al. [99] introduced this algorithm that solves optimization problems efficiently by imitating the swarming and aggregation behaviours observed in salp colonies.

In the mathematical model of salp chain swarming behaviour, we start with initializing the population randomly as in other optimization algorithms. The population is

broken up into two classes: leaders and followers. The former is always at the front of the salp chain, guiding the swarm to food sources, while the latter follow it and each other [99]. Each swarm aims for the target food source named "FS" which is located in the search space, and updates the leader salp's position according to FS as shown in Equation (2.29).

$$x_i^1 = \begin{cases} FS_i + y_1 \left((UB_i - LB_i) y_2 + LB_i \right), & y_3 \geq 0 \\ FS_i - y_1 \left((UB_i - LB_i) y_2 + LB_i \right), & y_3 < 0 \end{cases} \tag{2.29}$$

where x_i^1 denotes the position of the leader salp in the ith dimension, LB_i and UB_i represent the lower and upper bounds, and y_1, y_2, and y_3 are random values. The coefficient y_1 balances exploration and exploitation and is defined using Equation (2.30):

$$y_1 = 2e^{-\left(\frac{4n}{N}\right)^2} \tag{2.30}$$

Here, n denotes the current iteration, and N denotes the total number of iterations. In this paper, we select $N = 100$. The values y_2 and y_3 are randomly selected from the uniform distribution in $[0, 1]$. Equation (2.31) updates the position of each follower salp, where it follows its leader to form a salp chain:

$$x_i^m = \frac{1}{2} \left(x_i^m + x_i^{m-1} \right), \quad i \geq 2 \tag{2.31}$$

where x_i^m denotes the position of the mth follower salp in the ith dimension.

2.7 Experimental Design

This section provides an overview of the experimental design used in this thesis. Each step of the process is explained in detail in the subsequent sections.

2.7.1 Data collection

Empirical data plays a vital role in developing, validating, and benchmarking software defect prediction (SDP) models. It provides the foundation for evaluating model performance under realistic conditions and for identifying potential limitations. Such data is typically sourced from three primary categories: industrial software systems, open-source projects, and academic/research-oriented systems. Each source presents unique trade-offs in terms of accessibility, cost, and applicability. In recent years, open-source datasets have become the primary resource for empirical validation.

The widespread adoption of open-source datasets in empirical research is largely attributed to their public accessibility, transparency, and community support. These datasets are freely available, eliminating licensing costs and usage restrictions typically associated with proprietary systems. Furthermore, open-source projects are often continuously updated and well-documented, making them an ideal resource for academic and applied research. Their availability enables researchers to perform reproducible experiments and facilitates comparisons across different studies.

In this thesis, a diverse set of open-source software datasets is utilized to empirically evaluate the proposed hybrid models. The datasets span multiple software domains, offering variability in project size, code complexity, and defect density. This diversity enhances the generalizability and robustness of the experimental results. Selecting datasets from different domains also helps ensure that the models are not biased toward the characteristics of a specific software system, thus improving their applicability to real-world scenarios.

To maintain a structured and transparent research methodology, this thesis outlines the complete data acquisition and preprocessing pipeline. This includes the source and format of each dataset, the steps taken for cleaning and preparing the data (such as handling missing values and standardizing features), and the extraction of relevant

software metrics and defect labels. Additionally, each dataset is described in detail, highlighting attributes such as the number of instances, feature count, defect ratios, and metric types, which are critical for interpreting model performance.

The use of open-source datasets also supports the replicability and comparability of this research. As these datasets are publicly accessible, other researchers can reproduce the experiments and validate the results under similar or extended conditions. This commitment to reproducibility aligns with contemporary standards in empirical software engineering and strengthens the reliability of the findings presented in this study.

2.7.2 Dataset Details

In this section, we briefly introduce existing publicly available and commonly used benchmark datasets used in this thesis.

Table 2.1: Comparison of Dataset Repositories Used in SDP Research

Characteristics	**NASA**	**AEEEM**	**PROMISE**
No. of projects	13	5	20+
Programming Language used	C, C++	Java	Java, C, C++, JavaScript
Metrics Categories	Halstead, McCabe, Lines of Code	Object-oriented, Process, Previous defects	Object-oriented, Traditional, Process
No. of metrics	20–40	61	15–60
Data Granularity	Module/Function Level	Class level	Class/Method level
Description	NASA Metrics Data Program `http://openscience.us/repo/`	Open-source projects for evaluating defect prediction models `http://bug.inf.usi.ch`	Open-source, proprietary, and academic projects `http://openscience.us/repo/`

NASA Dataset

The NASA MDP collection comprises 13 distinct software projects, each representing different NASA missions, subsystems, or operational software components. The projects are typically identified by alphanumeric codes that preserve anonymity while maintaining research utility [100]. These projects vary in size, complexity, and application domain, offering diverse characteristics suitable for evaluating predictive models. Each dataset includes static code metrics such as LOC, cyclomatic complexity, Halstead metrics, and McCabe metrics, typically ranging from 20 to 40 features per project. These metrics are derived from static analysis of source code and are used as input features for defect prediction models.

A key characteristic of the NASA datasets is their class imbalance. The defective ratio varies significantly across projects - ranging from around 15% to 30% - which presents challenges for standard classification models and often necessitates resampling.

Table 2.2 provides basic statistical information for all the projects across NASA datasets utilized in this thesis, including the number of instances, attributes, defects, defective ratio, and imbalance ratio.

Table 2.2: Statistical Summary of NASA Datasets

Dataset	No. of Instances	No. of Attributes	Defective Instances	Non-defective Instances	Defective Ratio (%)	Imbalance Ratio
JM1	7783	22	1672	6111	21.48	3.65
MC1	1988	38	46	1942	2.31	42.22
KC1	2109	22	326	1783	15.46	5.47
PC3	1077	37	134	943	12.44	7.04
PC4	1458	37	178	1280	12.21	7.19
PC5	1711	38	471	1240	27.53	2.63
CM1	498	22	49	449	9.83	9.16
MW1	403	38	31	372	7.69	12.00

AEEEM Dataset

The AEEEM dataset repository consists of five widely studied open-source software projects: Apache Lucene (LC), Equinox (EQ), Eclipse JDT Core (JDT), Eclipse PDE UI (PDE), and Mylyn (ML), collected by D'Ambros et al. [101]. Each dataset is constructed at the file level and annotated with a binary defect label of 1 for defective

files and 0 for non-defective files based on post-release bug reports and commit logs. Each project includes 61 features, covering 17 source code metrics, five previous-defect metrics, five entropy-of-change metrics, 17 entropy-of-source code metrics, and 17 churn-of-source code metrics [102]. The number of instances ranges from 324 (EQ) to 1862 (Mylyn), and defect ratios vary significantly - from 9% in Lucene to 40% in EQ. This variation in class imbalance and metric diversity makes the AEEEM datasets particularly suitable for evaluating the generalizability and robustness of defect prediction models. Table 2.3 provides basic statistical information for all the projects in the AEEEM datasets.

Table 2.3: Statistical Summary of AEEEM Datasets

Dataset	No. of Instances	No. of Attributes	Defective Instances	Non-defective Instances	Defective Ratio (%)	Imbalance Ratio
Eclipse JDT Core	997	61	206	791	20.66	3.84
Eclipse PDE	1492	61	209	1283	14.01	6.14
Equinox	324	61	129	195	39.81	1.51
Lucene	691	61	64	627	9.26	9.80
Mylyn	1862	61	245	1617	13.16	6.60

This dataset represents a comprehensive benchmark collection for software defect prediction research, due to its high quality, rich metric set, and relevance to real-world software maintenance challenges.

PROMISE Dataset

The PROMISE repository (Predictive Modelling in Software Engineering) is a well-established and extensively used collection of software engineering datasets, curated by Jureczko and Madeyski [103] to support empirical software engineering research. It includes over 20 projects from both open-source and industrial domains, such as Apache Ant (build automation), Apache Camel (integration framework), Apache Ivy (dependency management), jEdit (text editor), Apache Log4j (logging framework), Apache Lucene (search engine), Apache POI (office document processing), Apache Synapse (enterprise service bus), Apache Velocity (template engine), and Apache Xerces (XML parser). These datasets are primarily derived from object-

oriented Java systems and are organized at the module or file level.

Each project in the PROMISE repository contains a diverse set of static code metrics, including size-based metrics (e.g., LOC), complexity metrics (e.g., McCabe cyclomatic complexity), CK and object-oriented metrics (e.g., coupling, cohesion, inheritance depth). Each project consists of 20 metrics, and these projects suffer from class imbalance, with the proportion of defective instances varying significantly across projects - from less than 5% to over 50% - posing a challenge for predictive modelling.

Table 2.4: Statistical Summary of PROMISE Datasets

Dataset	Instances	Attributes	Defective Instances	Non-defective Instances	Defective Ratio (%)	Imbalance Ratio
ant-1.7	745	20	166	579	22.28	3.49
camel-1.6	965	20	188	777	19.48	4.13
jedit-4.3	492	20	11	481	2.24	43.73
poi-3.0	442	20	281	161	63.57	0.57
Xalan-2.7	909	20	898	11	98.79	0.01

2.7.3 Data Preprocessing

Data pre-processing is a critical step in machine learning pipelines, significantly influencing the performance and reliability of machine learning models. The quality of input data directly impacts the effectiveness of predictive algorithms, making systematic data preparation essential for achieving robust and generalizable results. Raw software datasets often contain inconsistencies, noise, and scale disparities that can negatively impact model performance. Therefore, before model training, datasets should be clean, consistent, and undergo pre-processing operations to make them suitable for model training. This thesis applies three key pre-processing techniques: handling missing values, removal of outliers, and data normalization.

Handling Missing Values: Missing values are a common issue in real-world software datasets due to incomplete entries, data collection errors, or corruption. The presence of null or undefined entries can compromise the model training process.

In this study, missing values were addressed using median imputation, wherein missing entries in a given metric column were replaced with the median value of the corresponding feature across all available instances. This method preserves the overall distribution of the data and prevents bias introduced by arbitrary replacements or row deletions.

Outlier Removal: Outliers are the extreme values that deviate significantly from other data points and can distort statistical analysis and model predictions. These outliers are critical and need to be handled. To mitigate their impact, outlier analysis was conducted using the box plot technique, a graphical tool that identifies values outside the interquartile range (IQR). Specifically, instances falling below $Q1 - 1.5 \times$ IQR or above $Q3 + 1.5 \times$ IQR were flagged as outliers. These anomalous data points were then treated using median imputation since deleting the outliers would lead to degradation of the dataset and a reduction in the available data volume.

Data Normalization: Data normalization represents a fundamental preprocessing step in ML applications for SDP. The process involves transforming numerical features to a common scale without distorting differences in the ranges of values. This transformation becomes essential when dealing with software metrics that exhibit vastly different scales and units of measurement that can significantly lead to biased or suboptimal model performance of ML algorithms [104, 105].

To mitigate this issue, this study applies min-max normalization, which rescales each feature to a uniform range of [0,1]. The transformation is defined by the equation:

$$x_{\text{normalized}} = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

Where x is the original feature value, and $x_{\min}, x_{\max}$ represent the minimum and maximum values of the feature, respectively. This process maintains the relative distribution of the data while ensuring that all features contribute equally to the learning

process. Normalization is especially important for algorithms such as Support Vector Machines (SVM), Artificial Neural Networks (ANN), and Convolutional Neural Networks (CNN)[106]. By applying min-max normalization across all software metrics, the study ensures consistent data scaling, leading to more reliable and interpretable results in the defect prediction models.

2.7.4 Feature Selection

Feature selection represents a critical preprocessing technique in software defect prediction that involves identifying and selecting the most relevant subset of features from the original dataset. This process addresses the curse of dimensionality, where high-dimensional feature spaces can negatively impact machine learning model performance through increased computational complexity, overfitting, and reduced interpretability. In software defect prediction contexts, datasets frequently contain numerous software metrics ranging from simple size measures to complex object-oriented design metrics, many of which may be redundant, irrelevant, or noisy.

Effective feature selection can improve model generalisation by reducing overfitting, enhance model interpretability by focusing on the most important predictive factors, and reduce data collection costs in practical deployment scenarios [87]. Additionally, feature selection helps identify the most influential software metrics for defect prediction, providing valuable insights for software quality assurance practices and development process improvements.

Feature selection techniques are generally categorised into three broad types: **Filter**, **Wrapper**, and **Embedded** methods.

Filter Methods

Filter methods evaluate feature relevance independently of any specific ML algorithm by analyzing intrinsic data properties (e.g., correlation, variance) and their relationships with the target variable. They apply statistical measures to rank features and select the top-ranked ones. The evaluation process occurs before model training, making filter methods computationally efficient and scalable to large datasets [107]. This thesis uses Information Gain, Chi-Square, and Relief-F methods as feature selection techniques.

Information Gain is a technique that determines the relevance of a given attribute for the class label prediction. It is an entropy-based concept that measures the uncertainty in the class label after the attribute value is observed [108]. It can be found using the equation below:

$$IG(y) = -\sum_{i}^{m} p(x_i) \log p(x_i) + p(y) \sum_{i}^{m} p(x_i|y) \log p(x_i|y) + p(y') \sum_{i}^{m} p(x_i|y') \log p(x_i|y')$$

where x_i denotes the i^{th} class (defective/non-defective), $p(x_i)$ denotes the probability of the i^{th} class, $p(y)$ and $p(y')$ signify the probabilities of the presence/absence of the attribute y, and $p(x_i|y)$ and $p(x_i|y')$ denote conditional probabilities.

Chi-Square is the goodness-of-fit test that assesses the independence of two variables [109]. It determines the most relevant features by considering the highest values for the chi-squared statistic of the class label, computed as:

$$\chi^2 = \sum \frac{(Obs_i - Exp_i)^2}{Exp_i}$$

Where Obs_i and Exp_i are the observed and expected values, higher values indicate a stronger dependence of the feature on the output variable.

Relief-F Method is an instance-based method that uses nearest neighbours to estimate attribute quality [110]. For each instance in a training set, two nearest neighbours are identified: one of the same class (HIT) and one of a different class (MISS). The importance is calculated as:

$$I(y) = P\left(y \mid \text{nearest instance}_{\text{different class}}\right) - P\left(y \mid \text{nearest instance}_{\text{same class}}\right)$$

where P is the probability and y denotes the value of an attribute. An important attribute effectively differentiates instances of different classes while remaining consistent within the same class cite ahmed2014improving.

Wrapper Methods

Wrapper methods assess the quality of various feature subsets using a specific ML algorithm. Unlike filter methods, wrappers measure feature usefulness by training and evaluating a classifier. Techniques include Recursive Feature Elimination (RFE), Sequential Feature Selection, Forward/Backward Selection, and Genetic Search. In this thesis, we apply **PSO** and **OBWOA** algorithms as wrapper methods, described in Section 2.6.

Embedded Methods

Embedded methods integrate feature selection directly into the model training process, performing feature selection as an inherent part of the learning algorithm. These methods achieve computational efficiency by combining model training and feature selection into a single optimization process, while potentially achieving better performance than filter methods by considering feature interactions. Techniques like the Lasso L1 regression technique, the Decision Tree Feature Importance, the Regu-

larised Logistic Regression, and the Random Forest Feature Ranking are examples of embedded methods.

Recently, hybrid approaches combining filter, wrapper, or embedded techniques have emerged to leverage their strengths. These methods explore the feature space more effectively and help identify globally optimal feature subsets [111].

2.7.5 Data Balancing

After the elimination of redundant and irrelevant attributes from the dataset, a notable imbalance was observed between the number of instances in the low-defect and high-defect classes, leading to class imbalance in the data. There is a significant disproportion in the number of non-defective software modules compared to defective ones, which tends to exhibit bias toward the majority class during training. This potentially leads to models that achieve high overall accuracy while failing to identify the minority class, thereby deteriorating the comprehensive accuracy of software defect prediction models. The severity of class imbalance in software defect datasets is often substantial, as discussed in Section 1.7.2. To address the issue of imbalance in software defect datasets, we develop unbiased predictive models using data sampling techniques. In this thesis, we have employed oversampling techniques - Random Oversampling [112] and Synthetic Minority Over-sampling Technique (SMOTE) [113].

Random Oversampling is a simple technique that aims to balance the dataset by replicating instances from the minority class until the desired class distribution is achieved. This approach ensures that classifiers receive sufficient examples of the minority class during training, thereby improving their ability to detect defective modules.

SMOTE is an advanced oversampling technique that generates synthetic instances

rather than merely duplicating existing ones. In this technique, a sample from the minority class identifies its k nearest neighbours (typically $k = 5$) and creates new samples by interpolating between the original instances and their neighbours. By adding synthetic samples to the minority class, SMOTE balances the data and reduces the bias that causes uneven prediction results. As a result, models developed using this balanced dataset are more capable of accurately identifying defective and non-defective modules, enhancing their predictive accuracy and dependability in practical software engineering scenarios. By employing this approach, we seek to address the challenges posed by data imbalance and support the advancement of reliable and robust software defect prediction models.

2.7.6 Predictive Model Development and Validation

This research aims to develop predictive models for Software Defect Prediction (SDP), to identify modules that are likely to be defective in upcoming or future software releases. These models are constructed using supervised machine learning techniques, which learn from historical data associated with previous versions of software projects. By analyzing the relationships between software metrics and defect labels, the models are trained to recognize patterns that are indicative of defect-proneness. The training dataset used for model construction consists of independent variables, representing various software metrics (e.g., complexity, size, coupling), and a dependent variable, which indicates whether a module is defective (1) or non-defective (0). The learning algorithm processes this input to generate a classification model capable of mapping metric patterns to defect outcomes.

Once the model is trained, its performance must be evaluated on unseen data to assess its predictive capability. During validation, only the independent variables are provided as input, and the model is expected to predict the corresponding defect

label. These predicted labels are then compared against the actual defect labels to measure the model's accuracy and effectiveness. To ensure a reliable and unbiased assessment, this thesis employs the 10-fold cross-validation method. This approach is well-regarded for offering a sound balance between bias and variance, particularly when datasets are moderate in size. In 10-fold cross-validation:

- The entire dataset is randomly partitioned into 10 equal subsets, known as folds.
- The model is trained and tested over 10 iterations:
  - In each iteration, nine folds are used for training, and the remaining one fold is used for testing.
  - Each fold serves as the test set exactly once.
- Performance metrics such as Accuracy, Precision, Recall, F1-score, AUC, and MCC are computed for each iteration.
- The average of these metrics across all folds is taken as the final performance estimate.

This iterative process ensures that each data point contributes to both training and testing, thereby minimizing evaluation bias. Moreover, it reduces variability in results and enhances the generalizability of the model. 10-fold cross-validation is a robust choice, as it is widely accepted in software engineering literature, for validating the defect prediction models developed in this study.

2.7.7 Performance Evaluation Metrics

Performance evaluation is a crucial phase in predictive modelling, as it establishes the efficacy, dependability, and generalizability of the models that are created. The

performance measures generally give insights into different aspects of predictive performance by examining the predictions made by each model against the actual data. The choice of appropriate performance measures determines how effectively we can assess a model's ability to identify defective software modules while minimizing false predictions. In SDP modelling, where datasets are typically imbalanced with fewer defective modules than non-defective ones, traditional accuracy metrics often provide misleading assessments of model performance. The evaluation of SDP models requires robust and balanced metrics that can handle class imbalance effectively, and this comprehensive evaluation approach should ensure that models perform well not just on the majority class but also demonstrate strong capability in identifying the minority class (defective modules). In this thesis, various performance metrics are utilized to evaluate the effectiveness of our SDP models like ROC-AUC (Receiver Operating Characteristic - Area Under the Curve), F-measure, G-mean and MCC (Matthews Correlation Coefficient). We describe each metric below in detail.

AUC (Area Under the Receiver Operating Characteristic Curve) is a robust measure particularly effective for imbalanced and noisy datasets[49, 114]. It is a threshold-independent statistic that indicates the model's ability to distinguish between the positive (defective) and negative (non-defective) classes. It examines the relationship between true positive rate (recall or sensitivity) and false positive rate and plots an ROC curve at various threshold settings. AUC values range from 0 to 1, where AUC = 1 indicates perfect discrimination, and AUC = 0.5 suggests no discriminative power. The higher the AUC value, the better the predictive model at predicting defective or non-defective modules.

F-measure (F1 score) combines recall and precision into a single measure and is a harmonic mean of the two, as shown in the equation below:

$$\text{F-measure} = 2 \times \frac{\mathit{Precision} \cdot \mathit{Recall}}{\mathit{Precision} + \mathit{Recall}}$$

Precision (Positive Predictive Value) represents a measure that determines correctly classified defective instances out of all the defective instances:

$$\text{Precision} = \frac{True_{pos}}{True_{pos} + False_{pos}}$$

Recall (True Positive Rate) represents a measure that determines correctly classified defective instances among all actual defective instances:

$$\text{TPR} = \frac{True_{pos}}{True_{pos} + False_{neg}}$$

G-mean represents a balanced measure that is computed using the geometric mean of sensitivity (recall) and specificity:

$$\text{G-mean} = \sqrt{Sensitivity \cdot Specificity}$$

Where specificity is calculated as:

$$\text{Specificity} = \frac{True_{neg}}{\text{True}_{neg} + False_{pos}}$$

A high G-mean indicates that the classifier is performing well on both classes, which is essential in scenarios where both false positives and false negatives are costly.

Matthews Correlation Coefficient (MCC) measures the correlation between predicted and actual instances. The equation below is used to calculate its value, which falls between [-1,1]. The value of -1 indicates total disagreement, 0 indicates random prediction, and 1 indicates perfect prediction:

$$\text{MCC} = \frac{(\mathit{True}_{pos} \times \mathit{True}_{neg}) - (\mathit{False}_{pos} \times \mathit{False}_{neg})}{\sqrt{(\mathit{True}_{pos} + \mathit{False}_{pos})(\mathit{True}_{pos} + \mathit{False}_{neg})(\mathit{True}_{neg} + \mathit{False}_{pos})(\mathit{True}_{neg} + \mathit{False}_{neg})}}$$

2.7.8 Statistical Analysis

Statistical tests are essential to obtain quantifiable and statistically verifiable results regarding the predictions produced by a predictive model. Differences in performance (accuracy, AUC, F1-score, etc.) in predictive modelling can be slight, especially when comparing different models or configurations. These differences could be misleading due to random chance, overfitting, or sample bias [49]. Statistical tests assist in determining whether these differences are statistically significant and provide the framework for quantifying confidence in results derived from empirical data, enhancing reproducibility and credibility. In this thesis, the Friedman and Wilcoxon signed-rank tests are used, as these are non-parametric tests. The wide use of non-parametric tests is supported in literature because they do not require any assumptions about data normality, unlike parametric tests [115].

Friedman Test

This non-parametric test detects differences in k algorithms across multiple datasets, providing a robust framework for algorithm ranking and comparison studies [116]. It is based on the assumption that the performance measures of techniques computed over different datasets are independent of each other. The Friedman test evaluates:

- **Null Hypothesis** (H_0): There exists no significant difference in the performance of different techniques.

- **Alternative Hypothesis** (H_1): There exists a significant difference in the performance of different techniques.

The Friedman statistic (χ^2_F) is computed using the following steps:

1. **Ranking Performance:** For each dataset, the performance of k techniques is sorted in descending order (best to worst). Techniques are assigned ranks within each dataset, with Rank 1 for the highest performing technique and the lowest rank for the worst performing technique. If techniques exhibit identical performance, they receive the mean rank of their positions.

2. **Total Rank Calculation:** The sum total of all the ranks for each technique across all N datasets is computed. Let $T_1, T_2, T_3, \ldots, T_k$ denote the total ranks for techniques 1 to k.

3. **Chi-Square Statistic Computation:** The Chi-Square Statistic is computed using the equation:

$$\chi^2 = \frac{12}{N \cdot k \cdot (k+1)} \left(\sum_{i=1}^{k} T_i^2 \right) - 3 \cdot N \cdot (k+1)$$

where:

- *N*: Number of datasets,
- *k*: Number of techniques,
- T_i: Total rank of the *i*-th technique.

The degree of freedom is $k - 1$. If the computed Friedman statistic exceeds the critical value (i.e., falls in the critical region), the null hypothesis is rejected. This indicates statistically significant differences in the performance of the techniques

being compared. In this thesis, the Friedman test is applied at a 95% confidence level ($\alpha = 0.05$).

Wilcoxon Signed Rank test: This test can be employed as a post-hoc analysis following a significant Friedman test result, or it can be used independently to conduct pairwise comparisons between two techniques. It is appropriate to apply this test when both the techniques have been evaluated on the same set of datasets [117]. It serves as the non-parametric alternative to the paired t-test and is particularly useful when the differences between paired observations are not normally distributed. The Wilcoxon test evaluates:

- **Null Hypothesis (H_0):** There does not exist a significant difference in the performance of the two techniques.
- **Alternative Hypothesis (H_1):** There exists a significant difference in the performance of two techniques.

The Wilcoxon signed-rank test statistic (W) is computed in the following steps:

1. **Calculate differences:** For each pair of performances of two techniques, compute the difference D_i,
2. **Exclude zero differences:** Remove any pairs where $D_i = 0$.
3. **Rank the Absolute Differences:** Take the absolute values for each pair of differences $|D_i|$, and assign ranks to them in ascending order. If there are ties in absolute difference values, assign the average of the ranks that those values would occupy.
4. **Assign sign to the ranks:** Assign the original sign of D_i to each rank:
    - If $D_i > 0$, the rank will be positive.

- If $D_i < 0$, the rank will be negative.

5. **Compute the Test Statistic W:** The sum of the positive ranks (W^+) and the negative ranks (W^-) is computed. The Wilcoxon statistic (Z) is usually defined as:

$$Z = \frac{Q - \frac{1}{4}n_r(n_r + 1)}{\sqrt{\frac{1}{24}n_r(n_r + 1)(2n_r + 1)}} \tag{2.32}$$

Where, $Q = \min(W^+, W^-)$ and n_r is the number of non-zero differences. If the calculated Z statistic falls within the critical region for a given significance level ($\alpha = 0.05$), the null hypothesis is rejected. This indicates that there is a statistically significant difference in the performance of the two techniques being compared.

Chapter 3

Systematic Literature Review

3.1 Introduction

Software defects, or bugs, affect the quality, reliability, and efficiency of software products. With the increasing size and complexity of modern systems, traditional testing alone becomes insufficient, making SDP vital for identifying defect-prone modules before testing begins. Researchers have attempted to forecast software defects by analysing the relationship between software metrics and defect proneness. To achieve accurate predictions, researchers have applied a variety of machine learning (ML) techniques like decision trees, random forests, support vector machines, and neural networks. Alongside, search-based techniques such as genetic algorithms, particle swarm optimization, and firefly algorithms have proven effective in handling noisy and incomplete data spaces through optimization. A growing focus is on hybrid approaches, which combine ML with search-based methods. These hybrid techniques integrate the predictive strength of ML with the global optimization capability of meta-heuristics, resulting in faster convergence, avoidance of local minima, and adaptability across diverse projects.

Recent studies show that although hybrid models often outperform standalone approaches, additional research is still needed to highlight their potential in improving SDP. Studying these previous works systematically and summarizing them to capture meaningful information is crucial. This review describes, evaluates and presents a comprehensive analysis of different hybridized techniques across multiple dimensions in the form of different research questions stated below:

- RQ1) What are the commonly used search-based and ML techniques in forming hybridized techniques for SDP?
- RQ2) What is the year-wise distribution of studies using hybridized techniques for SDP?
- RQ3) Which prominent journals and conferences have published studies using hybridized techniques for SDP?
- RQ4) Which datasets and techniques are used in the experimental design of hybridized techniques for SDP?
  - RQ4.1) Which datasets are used?
  - RQ4.2) Which software metrics are used as independent variables?
  - RQ4.3) What are the feature selection techniques used?
  - RQ4.4) What are the parameter tuning techniques used?
  - RQ4.5) Which performance measures are used?
  - RQ4.6) Which validation techniques are used?
- RQ5) Which fitness functions are employed in search based techniques (part of hybridized techniques)
- RQ6) What is the performance of hybridized techniques for SDP?

  - RQ6.1) What is the overall performance of hybridized techniques?
  - RQ6.2) What is the comparative performance of hybridized techniques with traditional ML and statistical techniques?

- RQ7) What are the different statistical tests used to study hybridized techniques in SDP?
- RQ8) What are the strengths and weaknesses of hybridized techniques for SDP?
- RQ9) What are the threats to validity in studies using hybridized techniques for SDP?

In this review, we explore different SDP models proposed in the literature and highlight the gaps that still remain for future exploration. The findings of this systematic analysis, provides future guidance to software researchers and the industrial community by presenting a body of knowledge that wants to use and implement new hybrid techniques for SDP modelling.

The paper is structured as follows: Section 3.2 describes the review methodology process followed in this paper. Section 3.3 states the review protocol. Section 3.4 presents the results in the form of answers to the research questions. Section 3.5 outlines the future directions of this work. The results of this chapter are published in [118].

3.2 Review Procedure

The systematic review is carried out as per Kitchenham and Charters '(2007) guidelines [119]. The three primary activities involved are - planning, conducting, and reporting the review. The first step determines the need for review, identifies the research questions to be addressed, the development of the review protocol and its evaluation.

The review methodology is driven by the research questions, as these are the objectives that a literature review is intended to address. The primary studies are explored on the basis of research questions developed at this step. The review protocol development consists of various stages. In the first stage, a search strategy involves identifying the search terms and choosing suitable sources for collecting the relevant studies. The next stage forms the inclusion and exclusion criteria that filter the candidate studies based on relevance. After that, quality assessment criteria are prepared to ensure the quality of each candidate's study. The last step involves the procedure for defining the data extraction and synthesis part of all the primary research studies collected in the previous step.

The next step is conducting the review, which requires executing the steps described in the review protocol. The search strategy is implemented, and the inclusion/exclusion criteria are applied to extract the relevant primary studies. Their data is collected, analyzed, and synthesized in the forms of tables, graphs, figures, boxplots, etc., to draw conclusions about the results. The last stage is reporting the results, where the answers to each research question are addressed. The review protocol established in our work is thoroughly evaluated by all three researchers independently. Any inconsistencies found were resolved after comprehensive discussions.

3.3 Review Protocol

The review protocol includes the search strategy, inclusion and exclusion criteria, and the quality criteria for assessing the collected candidate studies. The following sections describe the review protocol followed.

3.3.1 Search Strategy

The review process involved selecting relevant and significant studies from the literature using the appropriate search terms. The preliminary search was initiated by scanning keywords such as “software defect prediction” Specific search keywords were formed by using the research questions and existing literature studies [120–122]. The synonyms and alternate terms for those keywords were concatenated using the Boolean expressions “OR”, “AND”, and “NOT” to form a complete search string.

Software AND (Defect OR Error OR Fault) AND (Prediction OR Proneness OR Tendency) AND (Hybrid OR Hybridized) AND (Search-Based OR Search Based OR Heuristic OR Meta-Heuristic) AND (Machine Learning OR Statistical) AND (Evolutionary OR Swarm Intelligence OR Bio-Inspired OR Optimization OR Genetic Algorithm OR Wolf Pack Algorithm OR Particle Swarm Optimization OR Cuckoo Search OR Ant Colony Optimization OR Firefly Search OR Artificial Bee Colony OR Genetic Programming OR Tabu Search OR Simulated Annealing OR Harmony Search OR Bat Search OR Evolutionary Programming OR Differential Evolution OR Artificial Immune System) AND (Decision Tree OR Support Vector Machine OR Naive Bayes OR Neural Network OR Logistic Regression OR Nearest Neighbor OR Random Forest).

Once the search string was formulated, it was required to select the most prominent and appropriate digital databases to extract relevant primary studies. The below digital libraries were chosen as per the author’s researcher expertise and criteria outlined by Chen et al. [123].

- IEEE Explore (https://ieeexplore.ieee.org).
- Springer (www.springer.com).
- Wiley Online Library (www.wiley.com).

- ACM Library (https://dl.acm.org).
- Google Scholar (https://scholar.google.com/).
- Science Direct (https://www.sciencedirect.com/).

Different online databases have different mechanisms for searching the papers, and we modified the search string accordingly. An initial search began by executing the search string, and we focused on the studies between January 2000 and December 2021. This list got reduced after discarding the studies based on titles, abstracts, and duplicate studies. After browsing through many articles, the inclusion and exclusion criteria were applied to select the most relevant studies. A final list of 72 primary studies was selected, satisfying the quality assessment criteria. We performed the snowballing procedure by scanning the references of the chosen studies and including them in our studies [124]. We found some review articles similar to our domain during our search process and extracted candidate studies from their reference sections. However, these review articles were not included as they don't possess empirical results.

3.3.2 Inclusion and Exclusion Criteria

The following inclusion and exclusion criteria were applied to the identified studies:

Inclusion Criteria

- Empirical studies use "defects" (defective/non-defective classes) as the independent variable.
- Empirical studies that use hybridized techniques for developing SDP models.
- Empirical studies that compare the performance of hybridized techniques with statistical or ML models.

- Empirical studies that apply hybridized techniques for feature selection or parameter optimization.

Exclusion Criteria

- Studies in which "defect count/number of defects" is the dependent variable.
- Studies in which there is no empirical evidence for SDP.
- Studies that do not discuss the application of any hybridized techniques in SDP models.
- Studies in which defect prediction models are based on expert judgment.
- Similar studies include those presented by the same author at a conference and an expanded version published in a journal. However, if the results in both studies were different, they were retained.
- Studies not in the English language.
- Review studies.

3.3.3 Quality Assessment Criteria

The quality evaluation score helps to exclude studies that may be essential for the literature review but are of lower quality than other studies. Low quality here signifies that the study scored low according to our set quality criteria even though the study is good and the results are valid. We formed a questionnaire consisting of 14 questions to evaluate the significance of each study according to the suggestions provided by Wen et al. [125]. For every study, a quality score of 0 (poor rating/No), 0.5 (adequate rating/Partly), or 1 (very good rating/Yes) was given for each question. Table 3.1

contains the list of all quality assessment questions and the total count of studies that were assigned 1, 0.5, and 0 scores for each question.

A total of 72 primary studies were chosen only after thoroughly applying the inclusion/exclusion criteria and meeting all quality standards. We analyzed the quality of the candidate studies based on 14 quality assessment questions mentioned in Table 3.1. A study can acquire the lowest score of 0 and the highest score of 14. We decided to keep those studies that scored equal to or above 50% of the quality criteria. Therefore, we accepted a primary study only if its quality score is greater than or equal to 7. The significance of a study on a particular topic is directly correlated with the quality score. Table 3.2 presents the lists of all primary studies and their quality scores. Each primary study is referred to by a unique identifier that will be used throughout the following sections of the paper.

Table 3.1: Quality Assessment of Reviewed Studies

S. No.	Quality Questions	Yes (1)	Partly (0.5)	No (0)
1	Whether the aims/objectives of the research are clearly stated?	35	37	0
2	Whether the study clearly states the defect prediction.	72	0	0
3	Whether the data collection procedure is clearly defined?	66	3	3
4	Whether the predictor variables clearly stated and defined?	41	13	18
5	Whether the study evaluation parameters clearly defined?	65	6	1
6	Whether validation techniques are clearly defined?	59	0	13
7	Whether the study clearly defines and describes the search-based techniques used.	59	8	5
8	Whether the study conduct any comparative analysis between search-based techniques, ML, or statistical techniques?	54	3	15
9	Whether fitness functions are clearly stated?	50	4	18
10	Whether the results and findings are reported clearly?	64	8	0
11	Whether any statistical tests are applied in the research to verify the results	33	0	39
12	Are the limitations of the review specified?	20	1	51
13	Whether the study is repeatable?	24	23	25
14	Whether the study's average citation count per year adequate?	56	0	16

Table 3.2: Primary Studies with Study Number, Quality Score, Reference, and Citation

Study#	QS	Reference	Citation	Study#	QS	Reference	Citation
HS1	9	(Khoshgoftaar et al. 2003)	[126]	HS37	7	(Niu et al. 2018)	[127]
HS2	11.5	(Gao et al. 2011)	[128]	HS38	12	(Geng 2018)	[129]
HS3	8	(Pendharkar 2010)	[130]	HS39	12.5	(Khari and Kumar 2018)	[131]
HS4	11	(Chiu 2011)	[132]	HS40	11.5	(Rhmann 2020)	[133]
HS5	13	(Sarro et al. 2012)	[134]	HS41	11.5	(Arora and Saha 2018)	[135]
HS6	11.5	(Yu 2012)	[136]	HS42	7	(Manjula and Florence 2018)	[53]
HS7	12	(Wahono and Suryana 2013)	[137]	HS43	11	(Malhotra et al. 2018)	[138]
HS8	10.5	(He et al. 2013)	[56]	HS44	7	(Manivasagam and Gunasundari 2018)	[139]
HS9	9.5	(Shuai et al. 2013)	[140]	HS45	8.5	(Brezočnik and Podgorelec 2019)	[141]
HS10	12.5	(Kayarvizhy et al. 2013)	[142]	HS46	10	(Cai et al. 2020)	[68]
HS11	11.5	(Wahono et al. 2014)	[54]	HS47	10.5	(Manjula and Florence 2019)	[143]
HS12	9	(Li et al. 2014)	[144]	HS48	10.5	(Tumar et al. 2020)	[145]
HS13	12.5	(Arar and Ayan 2015))	[146]	HS49	7.5	(Mabayoje et al. 2019)	[147]

Study#	QS	Reference	Citation	Study#	QS	Reference	Citation
HS14	11.5	(Jin and Jin 2015)	[18]	HS50	8.5	(Ayon 2019)	[148]
HS15	7.5	(Dhanalaxmi et al. 2015)	[149]	HS51	14	(Ni et al. 2019)	[150]
HS16	13	(Xia et al. 2016)	[151]	HS52	8.5	(Matloob et al. 2019)	[152]
HS17	14	(Ryu and Baik 2016)	[153]	HS53	10.5	(Arora and Saha 2019)	[154]
HS18	10.5	(Afzal and Torkar 2016)	[155]	HS54	11	(Alsghaier and Akour 2020)	[156]
HS19	13	(Hosseini et al., 2016)	[157]	HS55	11.5	(Thaher and Arman 2020)	[158]
HS20	14	(Shukla et al., 2018)	[159]	HS56	12	(Rhmann et al., 2020)	[160]
HS21	10	(Li et al., 2016)	[64]	HS57	12	(Zhang et al., 2020)	[161]
HS22	8	(Fazel 2016)	[162]	HS58	13	(Alsghaier and Akour 2021)	[67]
HS23	12	(Kumar and Rath 2017)	[163]	HS59	8	(Oloduowo et al., 2020)	[164]
HS24	9	(Jin and Dong 2016)	[165]	HS60	8.5	(Bahaweres et al., 2020)	[166]
HS25	7.5	(Jayaraj and Raman, 2016)	[167]	HS61	10.5	(Banga et al., 2020)	[168]
HS26	9.5	(Ibrahim et al., 2017)	[169]	HS62	10.5	(Yang et al., 2020)	[57]
HS27	11	(Moussa and Azar, 2017a)	[170]	HS63	7.5	(Balogun, Basri, Jadid, et al., 2020)	[171]
HS28	13	(Hosseini et al., 2018)	[172]	HS64	8	(Wu et al., 2020)	[173]

Study#	QS	Reference	Citation	Study#	QS	Reference	Citation
HS29	8.5	(Haveri and Suresh, 2018)	[174]	HS65	12.5	(Hassouneh et al., 2021a)	[55]
HS30	10	(Anbu and Anandha Mala, 2019)	[175]	HS66	12	(Kassaymeh et al., 2021)	[65]
HS31	8	(Rong and Cui, 2017)	[176]	HS67	7	(Malhotra et al., 2021)	[177]
HS32	10	(Pal et al., 2017)	[178]	HS68	11.5	(Zhu et al., 2021a)	[62]
HS33	8.5	(Thangavel and Pugazendi, 2017)	[179]	HS69	11.5	(Zhang et al., 2021)	[180]
HS34	12.5	(Turabieh et al., 2019a)	[63]	HS70	11	(Kang et al., 2021)	[181]
HS35	9	(Cao et al., 2018)	[182]	HS71	11	(Jin, 2021)	[183]
HS36	13.5	(Rhmann, 2018)	[184]	HS72	9	(Z. Li et al., 2021)	[185]

3.4 Review Results

3.4.1 Results specific to RQ1

RQ1. What are the commonly used search-based and ML techniques in forming hybridized techniques for SDP?

Over the years, researchers have employed various techniques for accurately predicting software defects. Initially, only statistical and ML techniques were used for SDP modeling. These techniques establish the relationship between software metrics and defect data. It became significantly more efficient with the introduction of search-based approaches. Hybrid approaches are now being employed, integrating ML/statistical and search-based techniques. Based on the primary studies selected for this systematic review, we grouped search-based

and ML techniques used to form hybrid techniques as follows.

- **Search-Based Techniques**
    1. Local Search
    2. Evolutionary
    3. Swarm Intelligence
    4. Bio-Inspired
    5. Nature-Inspired
- **ML Techniques**
    1. Decision Trees
    2. Ensemble Learners
    3. Support Vector Machines
    4. Neural Networks
    5. Other Techniques

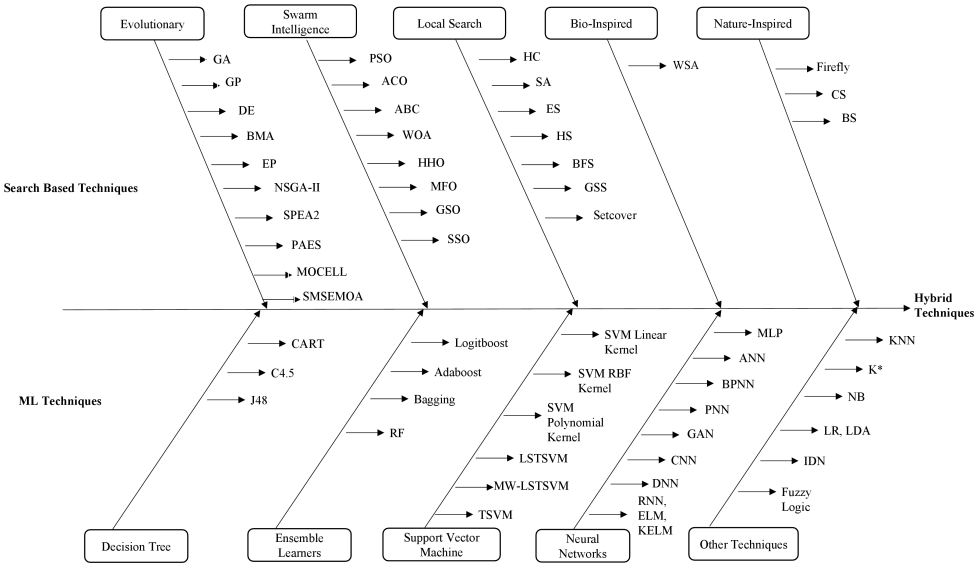

Figure 3.1: Classification of Search-based and ML techniques

Figure 3.1 describes the taxonomy of different search-based and ML techniques used in hybridized techniques for SDP. The table presenting hybridized techniques with respect to study numbers is provided in Appendix A.1. Figure 3.2 shows the percentage of studies that utilize specific categories of search-based and ML techniques in hybridized approaches for defect prediction.

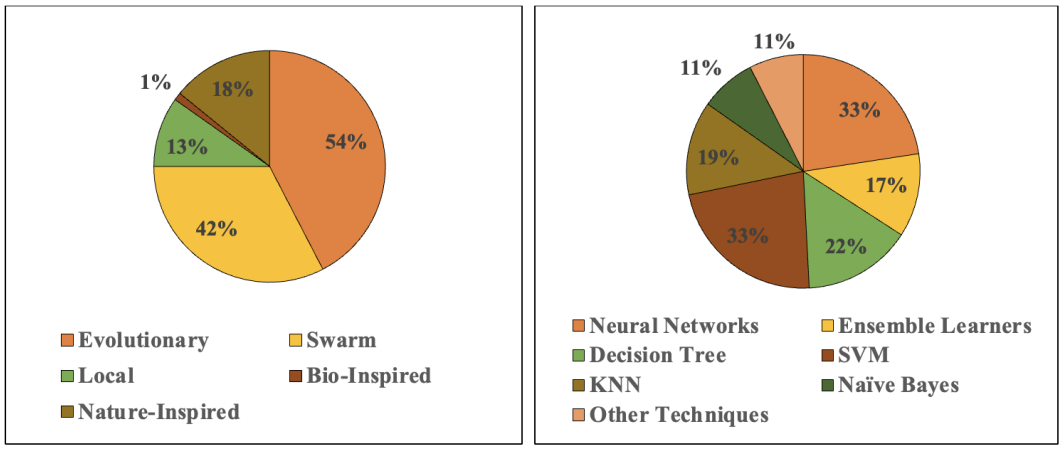

Figure 3.2: Distribution of studies by search-based and ML techniques used

Figure 3.2a illustrates that *evolutionary algorithms* are the most frequently used category within search-based techniques, appearing in 54% of the primary studies. These are used to form hybrid models such as GA-ANN, GP-NB, DE-RF, GA-ELM, among others. Genetic Algorithm (GA) is the most popular within this category. *Swarm intelligence algorithms* follow, used in 42% of the studies, forming hybrids such as PSO-LDA, ACO-LRNN, ABC-ANN, and WOA-KNN, with Particle Swarm Optimization (PSO) being the most commonly adopted. *Nature-inspired*, *local*, and *bio-inspired* techniques account for 18%, 13%, and 1% of the studies respectively, forming combinations like Firefly-SVM, HS-Bagging-DT, and WSO-SVM.

Figure 3.2b indicates that Support Vector Machines (SVM) and Neural Networks are the most popular ML algorithms, each used in 33% of the studies. Decision Tree algorithms are found in 24% of studies, while K-Nearest Neighbors (KNN), Ensemble Learners, and NaÃ¯ve Bayes appear in 19%, 17%, and 11% respectively. Other ML techniques such as K*, Instance-based Distance Networks (IDN), Fuzzy Mutual Information, Fuzzy Logic, and Rule-Based Learning account for 11% of the studies. Statistical techniques, including Logistic Regression (LR) and Linear Discriminant Analysis (LDA), are also employed in hybrid

models, comprising 17% of the studies. Notably, there is a growing interest in deep learning models like Convolutional Neural Networks (CNNs) and Deep Neural Networks (DNNs) in the software engineering domain in recent years, due to their ability to process large-scale data and model complex relationships among variables to enhance prediction performance [186, 187].

Hybrid techniques have gained popularity in predictive modeling due to their ability to combine the global optimality of search-based approaches with the fast convergence of ML techniques [17]. These methods also show robustness in handling noise and data uncertainty [188]. They often follow a simplified automated search process with fewer training parameters, enhancing the quality of solutions [142, 189]. As observed, diverse variants within each category of algorithms have been combined to develop effective hybrid models, which are detailed in Appendix A.1.

3.4.2 Results specific to RQ2

RQ2. What is the year-wise distribution of studies using hybridized techniques for SDP?

The primary studies are classified based on the year and type of publication, as depicted in Figure 3.3. It shows that before 2013, only six studies had applied hybrid techniques in SDP. The majority of primary studies - approximately 92% of the total - were published between 2013 and 2021.

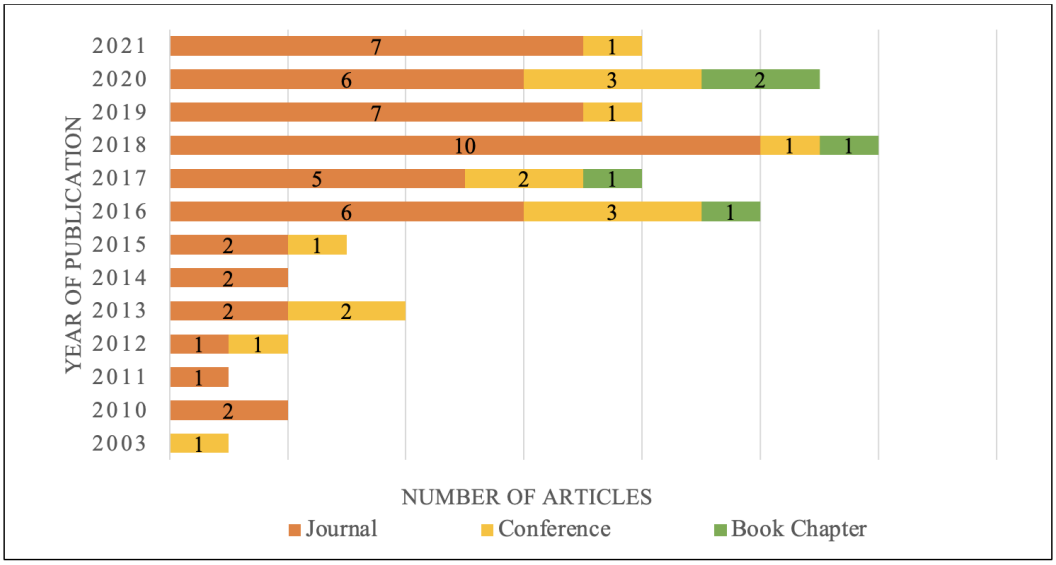

Figure 3.3: Classification of Primary studies by year and publication

The term "Search-Based Software Engineering" (SBSE) was first introduced by Harman and Jones in 2001 [14]. Later, in 2010, Harman [190] emphasized the applicability of meta-heuristic search techniques in the software engineering domain. Since then, the interest of both researchers and practitioners has grown significantly, leading to extensive discussions, reviews, and identification of critical challenges posed by SBSE to modern software engineering. A noticeable rise in the number of studies occurred after 2016, primarily due to the emergence of deep learning models being combined with search-based techniques in the SDP domain. This relatively recent approach has demonstrated improved performance compared to traditional machine learning and statistical techniques when integrated with search-based optimization.

3.4.3 Results specific to RQ3

RQ3: Which prominent journals and conferences have published studies using hybridized techniques for SDP?

This section describes the publication sources of the primary studies that utilize hybridized

techniques for developing Software Defect Prediction (SDP) models. According to Figure 3.4, 71% of the primary studies were published in journals, 22% in conferences, and 7% in book chapters. Thus, the number of studies published in journals (51) is significantly higher than those in conferences (16) and book chapters (5).

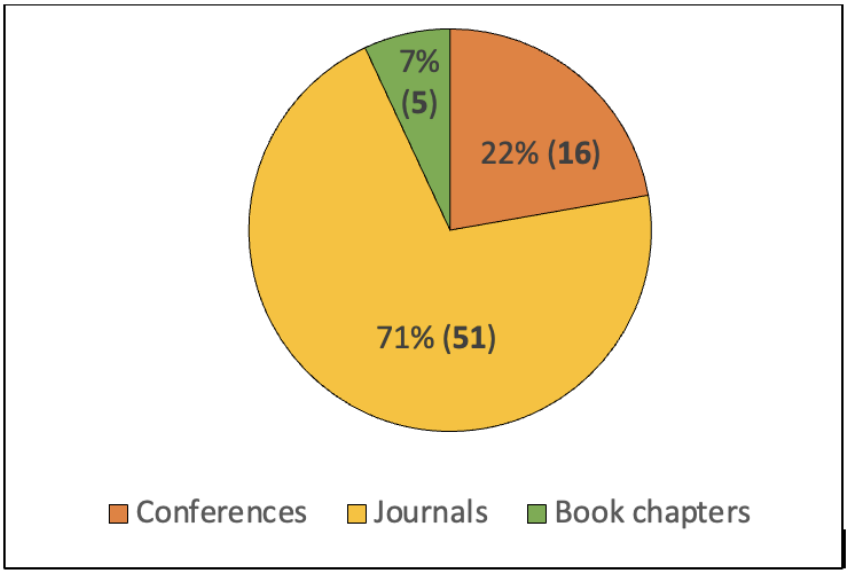

Figure 3.4: Distribution of primary studies according to their type

Table 3.3 describes the details of the prominent journals and conferences, with their impact factors (current date) and the count of studies in each publication. These journals and conferences are influential, high in impact factor, and contain relevant previously published articles related to software engineering [191, 192]. Three papers have been published in the *Journal of Systems and Software* (HS27, HS51, HS68), *Software - Practice and Experience* (HS2, HS54, HS58), *Applied Soft Computing* (HS13, HS14, HS17), and *Expert Systems with Applications* (HS4, HS34, HS71). One journal paper is published in *IEEE Transactions on Software Engineering* (HS16) and *Information and Software Technology* (HS28). All other research articles are also from other well-known publications. The top five highly cited studies are HS2, HS16, HS13, HS46, and HS1. We also list the top fifteen highly cited studies in

Appendix A.2.

Table 3.3: Publication Distribution by Type and Impact Factor

Publication Name	**Type**	**Number of Studies**	**Impact Factor**
Journal of Systems and Software	Journal	3	2.829
Software- Practice and Experience	Journal	3	2.028
Applied Soft Computing	Journal	3	6.725
Expert Systems with Applications	Journal	3	6.954
Information Sciences	Journal	2	6.795
Information and Software Technology	Journal	1	2.73
IEEE Transactions on Software Engineering	Journal	1	3.331
IEEE Int. Conf. on Tools with AI	Conference	1	-
ACM Symposium on Applied Computing	Conference	1	-
Int. Conf. on Predictive Models and Data Analytics in SE	Conference	1	-

3.4.4 Results specific to RQ4

RQ4: Which datasets and techniques are used in the experimental design of hybridized techniques for SDP?

This section determines the most commonly used datasets, software metrics, feature selection techniques, parameter tuning techniques, validation methods, and performance measures used to develop an SDP model using hybridized techniques.

RQ4.1: Which datasets are used?

A variety of datasets are used in primary studies that apply hybridized techniques for SDP models. There are two kinds of datasets - public and private. Public datasets are publically available for free, whereas private datasets are private in nature and hence not open to the public easily. Figure 3.5 represents the percentage of studies against the different types of datasets used. Datasets of our domain can be categorized as below:

1. **PROMISE Repository Dataset:** This kind of dataset is available for free and accessible in the PROMISE repository. About 38% of the selected studies (HS3, HS5, HS10, HS16, HS17, HS18, HS19, HS20, HS23, HS27, HS28, HS29, HS34, HS39, HS43, HS45, HS48, HS51, HS54, HS55, HS58, HS63, HS65, HS66, HS68, HS69, HS71) use this kind of dataset. Java datasets like Apache POI, Apache Xalan, jEdit version, etc., are all available in the PROMISE repository [103].

2. **NASA Dataset:** The most commonly used datasets for SDP come from the NASA Metrics Data Program, providing open-source software data to the public. About 64% of the studies (HS3, HS4, HS6, HS7, HS8, HS9, HS11, HS15, HS12, HS13, HS14, HS18, HS21, HS22, HS24, HS25, HS26, HS30, HS31, HS32, HS33, HS35, HS36, HS37, HS38, HS40, HS41, HS42, HS44, HS45, HS46, HS47, HS49, HS50, HS52, HS53, HS54, HS57, HS58, HS59, HS60, HS61, HS63, HS64, HS66, HS69) include NASA datasets for model development. JM1, KC1, PC1, MW1, etc., all these datasets are available in a public NASA repository [193].

3. **Open-source Software Datasets:** Repositories like GitHub and Sourceforge.net provide open-source software projects from which researchers can download free data for their use. Eight studies (HS56, HS62, HS63, HS67, HS68, HS70, HS71, HS72) acquire data from open-source libraries. Some commonly used open-source benchmark datasets include AEEEM [194] and ReLink [195].

4. **Academic Dataset:** This dataset includes student research projects developed as a part of their course or degree. Only one study, HS17, has used 17 academic projects as a single merged academic dataset in their research.

5. **Industrial Dataset:** This kind of dataset is proprietary in nature and belongs to some industries or private organizations, like the telecommunication dataset. About 7% of the total studies include industrial datasets.

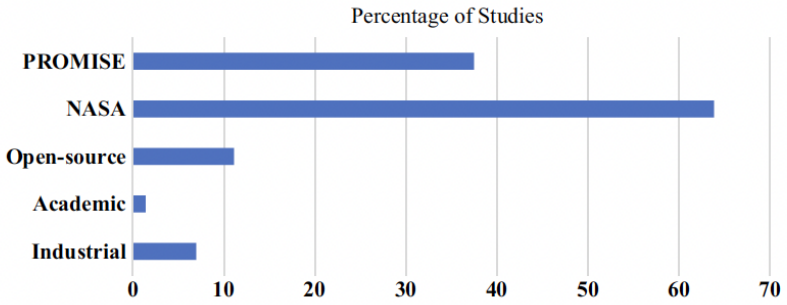

Figure 3.5: Distribution of studies according to the type of datasets

RQ4.2: Which software metrics are used as independent variables?

Software metrics quantitatively analyze and describe the characteristics of the software projects like size, complexity cohesion, coupling, inheritance, etc. These metrics act as significant predictors or independent variables in defect prediction modeling. We define the classification based on the metrics used in the primary studies that apply hybridized techniques. Figure 3.7 depicts the percentage of studies according to the metrics used.

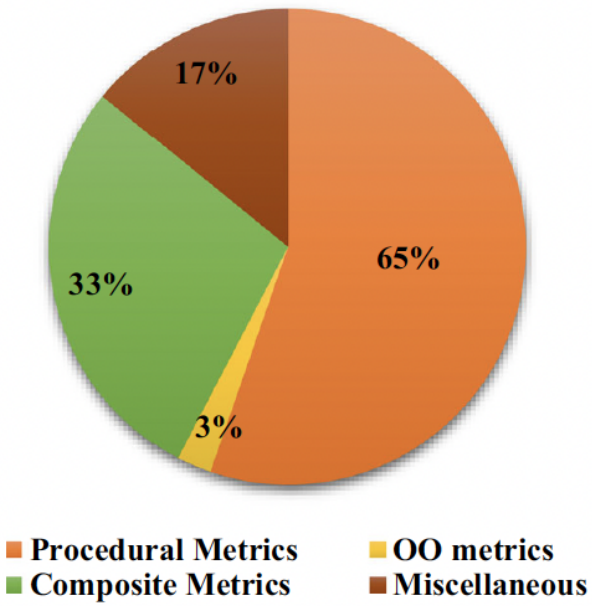

Figure 3.6: Percentage of studies according to software metrics

1. **Traditional/Procedural Metrics -** These are also known as static code metrics as they measure software attributes extracted directly from the source code without requiring any execution. Metrics like McCabe and Butler (1989)[196] , Mccabe (1976) (cyclomatic complexity, essential complexity, etc)[43], Halstead (1977) (program length, volume, effort, etc.) [197] and size metrics like LOC have been used by about 65% of the primary studies namely HS3, HS4, HS6, HS7, HS8, HS9, HS11, HS12, HS13, HS14, HS15, HS18, HS21, HS22, HS24, HS25, HS26, HS27, HS30, HS31, HS32, HS33, HS35, HS36, HS37, HS38, HS40, HS41, HS42, HS44, HS46, HS47, HS49, HS50, HS52, HS53, HS54, HS57, HS58, HS59, HS60, HS61, HS63, HS64, HS66, HS69.

2. **Object-Oriented Metrics -** OO metrics could capture various key aspects of the software that static code metrics couldn't. These metrics could measure OO paradigms

like - encapsulation, inheritance, coupling, cohesion, polymorphism, etc. Various researchers also consider the size metric LOC an OO metric; hence we have included this metric in both categories (static and OO). About 3% of the total studies, i.e., only 2, use only OO metrics, namely HS39 and HS67.

3. **Composite Metrics -** OO and static metrics, together as independent variables, have been grouped in this category of composite metrics. 33% of the total studies (HS5, HS10, HS16, HS17, HS18, HS19, HS20, HS23, HS28, HS29, HS34, HS43, HS45, HS48, HS51, HS54, HS55, HS58, HS63, HS65, HS66, HS68, HS69, HS71) use combination of both OO and static metrics together for defect prediction.

4. **Miscellaneous Metrics -** Studies that do not use OO or static metrics fall under this category. 12 primary studies use different types of metrics - HS56, HS62, and HS70 include change metrics; HS1 includes one LOC-based process metric and 4 product metrics; HS2 includes 24 product metrics; HS51, HS68, and HS71 use a combination of complexity and size metrics; and HS63, HS68, and HS71 use 61 variety of metrics like - defect metrics, the entropy of change, source code metrics, etc. HS72 includes metrics related to procedure information, dataflow information, and other code coverage metrics.

RQ4.3 What are the feature selection techniques used?

Feature selection is a data pre-processing technique that reduces the original features set to form a smaller optimal subset by eliminating redundant and irrelevant features. It helps remove noise in the dataset and increases the efficiency of a predictive model. There are three feature selection techniques - Filter, Wrapper, and Embedded methods. Filter methods measure the relevance of the attributes by finding a correlation with dependent variables, whereas wrapper methods measure the usefulness of attributes by training a model on them. Search-based techniques act as wrapper feature selection methods that require a learning algorithm to select

optimal attributes of a dataset. Embedded methods combine the features of both filter and wrapper methods. Table 3.4 and Table 3.5 show the different feature selection techniques used by primary studies.

Table 3.4: Filter feature selection techniques

Primary studies	**Filter techniques**	**Count of studies**
HS13, HS18, HS26	Correlation-Based Feature Selection	3
HS2, HS18, HS28, HS53	Information Gain	4
HS2, HS53	Chi-Square Statistic	2
HS2, HS53	Gain Ratio	2
HS2	Kolmogorov–Smirnov Statistic	1
HS2, HS18, HS53	Relief Algorithm	3
HS2, HS53	Symmetrical Uncertainty	2
HS18	Principal Component Analysis	1
HS57	Kernel Principal Component Analysis	1
HS18	Consistency-Based Subset Evaluation	1
HS53	Probabilistic Significance	1
HS33	Mutual Information	1

Among 72 primary studies, 46% of studies use feature selection techniques in developing SDP models. In Table 3.4, we can observe that Information Gain is the most frequently used filter technique. Information gain for each attribute is calculated, and those with a higher value are selected. In Table 3.5, we can observe that GA is the more commonly used wrapper technique. It has a strong ability to perform a global search and explore the search space using various crossover techniques [53, 148, 198]. Several authors [150, 152, 154] have also compared different search-based techniques with other filter methods (Information Gain, Gain Ratio, Correlation method) and wrapper (best first search, greedy stepwise, linear forward selection). It is concluded that search-based techniques outperform the conventional feature selection methods because of their stochastic nature, efficient search space exploration, and avoidance of getting trapped in local optima.

Table 3.5: Wrapper feature selection techniques

Primary studies	Wrapper techniques	Count of studies
HS2	ES, HS, AHS	1
HS7, HS11, HS14, HS27, HS34, HS36, HS38, HS52, HS53, HS54, HS61, HS63, HS67	PSO	13
HS11, HS22, HS23, HS27, HS32, HS34, HS36, HS42, HS47, HS49, HS50, HS52, HS53, HS54, HS56, HS58, HS60, HS61, HS63	GA	19
HS18	GP	1
HS26, HS63	Bat Search	2
HS30, HS53, HS59, HS63	Firefly Search	4
HS34, HS44, HS63	ACO	3
HS49	Multi-Objective Evolutionary Search	1
HS48	Binary Moth Flame Optimization	1
HS58, HS65, HS68	Whale Optimization	3
HS59	Wolf Search	1
HS55	Harris Hawks Optimization	1
HS51, HS63, HS69	MOFES-NSGA-II, NSGA-III	3
HS68	Simulated Annealing	1
HS63	Bee Colony Optimization	1
HS63	Cuckoo Search Optimization	1
HS63	Elephant Search	1
HS63	Flower Search	1
HS63	Rhinoceros Search	1

RQ4.4: What are the parameter tuning techniques used?

The values of parameters in a model control its learning process and influence its performance. When building hybrid-based models, the optimal configuration/parameter settings of classification algorithms must be determined to achieve the optimum classification performance of the model. Since search-based techniques explore different possible configurations in their search space, they are best suitable for this problem. Table 3.6 represents search-based techniques found in the primary studies used for parameter optimization of different

classification algorithms.

Table 3.6: Parameter optimization techniques

Primary studies	**Techniques used for parameter optimization**	**Count of studies**
HS4, HS8, HS10, HS36, HS41, HS71, HS72	PSO	6
HS5, HS16, HS29, HS36, HS41, HS57	GA	6
HS13	ABC	1
HS21, HS64	Bat Search	2
HS31, HS35, HS37, HS46	Multi-Objective Cuckoo Search	4
HS33, HS41	Firefly Search Algorithm	2
HS43, HS62	Differential Evolution	2
HS32	Bird Mating	1
HS66	Salp Swarm Optimization	1
HS70	Harmony Search	1
HS43	Simulated Annealing	1
HS72	Wolf Pack Algorithm	1

GA (HS5), PSO (HS8), and Bat algorithm (HS21), MOCS (HS31, HS35, HS37, HS46), Firefly Algorithm (HS33) are used to optimize parameters of SVM. PSO (HS4, HS10) is used to obtain connection weights of integrated decision networks and ANN, respectively. ABC (HS13), GA (HS29, HS41), PSO (HS41), and Bird Mating Algorithm (HS32) are used to optimize weights for ANN. GA (HS16) is used to optimize source project weights and build a GA classifier. DE and SA (HS43) for optimizing parameters of CART and Random Forest. GA (HS57) is used to optimize the weights and bias of the ELM algorithm. DE (HS62) determines the coefficients of the LR model according to the metric Density percentile average. An improved version of PSO (HS71) is used for parameter tuning of Kernel Twin SVM. A multi-objective Bat algorithm (HS64) addresses the problem of imbalance and synchronous parameter optimization of SVM. A hybrid of the Wolf Pack algorithm and PSO (HS72) optimizes the hyperparameters of the DNN.

RQ4.5: Which performance measures are used?

The predictive capability of various SDP models developed using hybridized techniques should be evaluated, as different authors consider different performance measures. Table 3.7 lists all the performance metrics along with the studies in which they are used.

Table 3.7: Performance measures used by primary studies

Primary studies	**Performance measures**
HS3, HS4, HS5, HS6, HS8, HS10, HS12, HS13, HS15, HS21, HS22, HS23, HS24, HS25, HS26, HS27, HS29, HS30, HS31, HS32, HS33, HS36, HS41, HS42, HS45, HS47, HS49, HS50, HS52, HS53, HS54, HS58, HS59, HS61, HS63, HS66	Accuracy
HS4, HS5, HS6, HS12, HS13, HS15, HS17, HS19, HS20, HS21, HS24, HS25, HS30, HS31, HS33, HS35, HS37, HS40, HS41, HS42, HS44, HS46, HS47, HS49, HS48, HS50, HS54, HS55, HS56, HS57, HS58, HS59, HS61, HS66, HS67, HS70, HS72	Recall (sensitivity)
HS4, HS5, HS12, HS19, HS21, HS24, HS25, HS30, HS31, HS33, HS39, HS40, HS43, HS44, HS47, HS49, HS50, HS54, HS56, HS57, HS58, HS59, HS61, HS72	Precision
HS2, HS6, HS7, HS9, HS11, HS13, HS14, HS17, HS18, HS26, HS32, HS34, HS38, HS47, HS48, HS51, HS52, HS55, HS60, HS63, HS65, HS66, HS67, HS68, HS71, HS72	AUC
HS13, HS17, HS31, HS33, HS35, HS37, HS66, HS59, HS69, HS70	Probability of false alarm (pf)
HS6, HS15, HS41, HS42, HS47, HS48, HS50, HS54, HS55, HS58, HS66	Specificity
HS4, HS5, HS12, HS16, HS19, HS20, HS21, HS23, HS24, HS25, HS28, HS30, HS31, HS39, HS43, HS44, HS45, HS47, HS50, HS52, HS54, HS57, HS58, HS59, HS61, HS67, HS68, HS69, HS71, HS72	F-measure
HS19, HS28, HS36, HS46, HS60, HS64, HS68, HS69	G-mean
HS50, HS52, HS68, HS69	Mathews correlation coefficient (MCC)
HS1, HS4, HS13, HS15, HS16, HS17, HS20, HS27, HS29, HS33, HS41, HS44, HS49, HS50, HS51, HS52, HS54, HS58, HS66, HS62, HS70	Miscellaneous

Figure 3.7 depicts the distribution of studies based on these performance metrics. The most commonly utilized performance metrics while evaluating hybrid-based SDP models

were recall and accuracy, which appeared in about 53% and 50% of the selected primary studies. F-measure was used to assess around 42% of the studies and AUC in about 36% of the studies. Accuracy though widely used is not recommended and may be misleading while evaluating the model's performance on imbalanced datasets [199]. Instead, the AUC measure is considered more effective while handling imbalanced and noisy datasets with respect to class distribution [200]. Researchers should employ a suitable performance metric that would produce valid and reliable results and improve the study's conclusion validity. Other studies use Precision (33%), Probability of False alarm (14%), Specificity (15%), G-mean (11%), and MCC (6%). Some miscellaneous metrics (29%) like Type I error, Type II error, Defect density, Balance, Error rate, Mean Absolute Error, False Positive rate, Misclassification rate, mean absolute relative error, R-square, Standard error of the mean, mean square error, Root mean square error, Hypervolume are also used in very few studies.

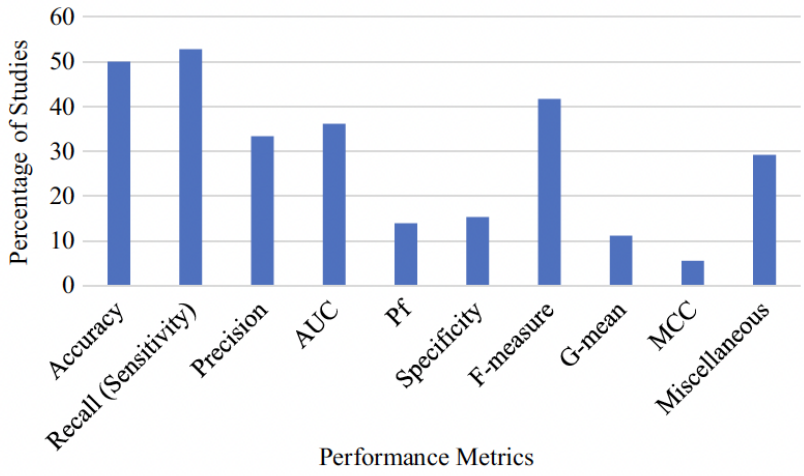

Figure 3.7: Distribution of primary studies based on performance metrics

RQ4.6: Which validation techniques are used?

An efficient validation method can increase the reliability of the results produced by a predictive model. Fifty-nine primary studies reported using validation methods categorized

as Hold-out, K-Fold cross-validation, and Cross-project validation. Table 3.8 contains all the validation methods utilized in the studies, a brief description of each, and the study identifiers that use a specific method. Figure 3.8 shows the study based on the validation methods used.

Table 3.8: Description of validation methods

Validation techniques	**Brief description**	**Study#**
K-Fold Cross-Validation	The dataset is split into K equal-sized partitions in this procedure. K-1 partitions are used to train the model in each iteration, whereas 1 out of K partitions is used to test the model. This procedure is repeated K times. The commonly used K values are 5 and 10	HS2, HS4, HS7, HS8, HS9, HS11, HS13, HS17, HS18, HS19, HS23, HS26, HS27, HS28, HS31, HS34, HS35, HS36, HS37, HS39, HS41, HS45, HS48, HS49, HS50, HS53, HS54, HS55, HS56, HS57, HS58, HS59, HS60, HS61, HS62, HS63, HS66, HS70
Hold-out Validation	The dataset is split randomly into two parts- one is used for training a model, and another is used for testing a model. The datasets are divided into specific ratios. The commonly used ratios are 80:20 and 70:30	HS1, HS3, HS5, HS6, HS10, HS12, HS14, HS21, HS22, HS24, HS32, HS33, HS34, HS38, HS40, HS51, HS52, HS65
Cross-project Validation	The model is trained using a dataset from a specific project (X) and is validated using a dataset from another project (Y)	HS16, HS20, HS69, HS71

We discovered that the most common validation approach is K-fold cross-validation, which gives the mean values of all the partitions of the training dataset, reducing variability. Unlike hold-out validation, the dataset is unaffected by the created partitions. But one disadvantage is that this method requires more computational cost as the model training needs to be done K times.

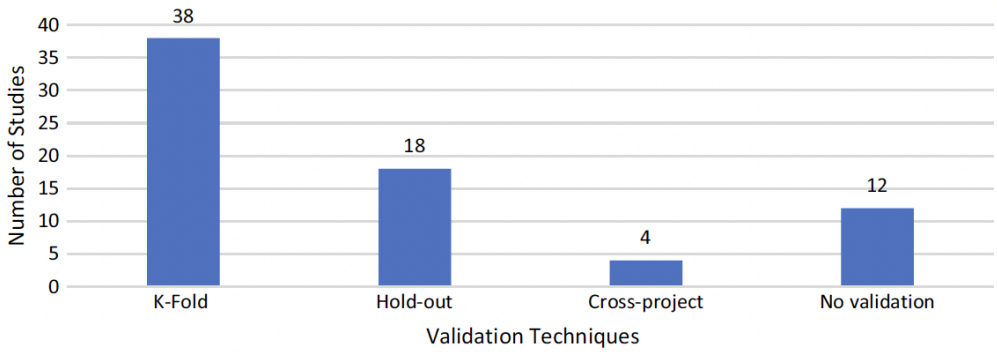

Figure 3.8: Distribution of studies based on validation methods used

3.4.5 Results specific to RQ5

RQ5: Which fitness functions are employed in search-based techniques (part of hybridized techniques) for SDP?

A population comprises multiple candidate solutions that are evaluated for goodness using a fitness function to arrive at an optimal or suboptimal solution. The fitness function should be able to quantify how well a given solution fits the problem. According to Harman and Clark [4], software metrics can act as fitness functions in search-based techniques and, therefore, can be used for the creation of software predictive models. Studies have employed various performance measures, such as fitness functions like accuracy, F-measure, etc., and their variants in searchbased techniques. Around 81% of the primary studies specify fitness functions explicitly. Table 3.9 lists the various fitness functions used in different studies.

According to Table 3.9, accuracy and its variants are the most commonly utilized fitness

Table 3.9: Fitness functions employed by search-based techniques in hybrid-based models

Fitness functions	**Study#**
Misclassification Cost	HS1, HS20, HS24, HS26
Accuracy	HS3, HS4, HS7, HS10, HS11, HS23, HS27, HS34, HS36, HS42, HS44, HS45, HS47, HS48, HS55, HS61, HS68
F-measure	HS5, HS16, HS30
Multi-objective: Specificity and Sensitivity	HS6
Mean absolute error	HS18, HS29
F1 Score * G-mean	HS19, HS28
Multi-Objective: Probability of detection (Pd) and probability of false alarm (pf)	HS17, HS20, HS31, HS35, HS37, HS46, HS64
AUC	HS51, HS65, HS66, HS60
Miscellaneous	HS8, HS9, HS12, HS13, HS14, HS15, HS17, HS21, HS22, HS32, HS38, HS50, HS51, HS57, HS62, HS67, HS69, HS70, HS71, HS72
Not Clearly stated	HS25, HS33, HS39, HS40, HS41, HS43, HS49, HS52, HS53, HS54, HS56, HS58, HS59, HS63

functions, accounting for 24% of all primary studies. Pd and pf are used as a multiobjective fitness function in 10% of the studies with the objective of maximizing recall and minimizing pf values. The misclassification cost and its variants are found in 6% of the studies. F-measure and its variants are used in 4% of the studies. AUC is considered an objective function in 6% of the total studies. Other fitness functions include G-mean, Balance, the sum of absolute error, mean square error, etc., which are considered miscellaneous functions. A total of 14 studies have not explicitly stated the fitness functions used in their studies.

However, it is necessary to explore different fitness functions as they form a key factor in any optimization technique. It influences the classification performance, computational time, and the input search space for solutions. The kind of learning algorithm also determines the performance of a fitness function [192]. Multiobjective fitness functions should also be studied because they balance multiple objective functions while giving a stable predictive model.

3.4.6 Results specific to RQ6

RQ6: What is the performance of hybridized techniques for SDP?

Various primary studies have proposed numerous hybridized techniques, and their performance must be evaluated to ensure their relevance in SDP modeling. Therefore, we present the predictive capabilities of these techniques by reporting the performance measures analyzed in RQ4.5. The results of only those techniques are recorded and examined on a minimum of three different data sets and used in at least two primary studies to generalize the findings. It is necessary to avoid bias caused by a technique's superior performance in research or on specific datasets.

RQ6.1: What is the overall performance of hybridized techniques used for SDP models?

We record the values of Accuracy, Recall, and F-measure for hybridized techniques on different datasets to examine their performance. These metrics were chosen for evaluation because they are the most frequently used performance metrics. We present the descriptive statistics of performance measures after removing the outliers in Table 3.10. The outliers were removed because some techniques may produce biased predictions when applied to specific datasets. Therefore, outlier analysis was essential and done with the help of boxplots for the performance measures dataset-wise. Figure 3.9 represents the recall outliers reported on different datasets. We can observe that, in total, there are nine recall outliers. MC2 is an outlier for GA-KNN and GA-NB algorithms with low recall values. Xalan-2.5, poi-2.5, poi-3.0, KC4, and Lucene-2.2 observe low recall values and are outliers for the PSO-SVM algorithm. PC2 and PC4 are outliers for PSO-ANN and GA-NB algorithms with higher recall values. Figure 3.10 represents the F-measure outliers reported on different datasets. According to the figure, GA-DT and GA-ANN have one outlier each for the Log4j dataset. Lucene-2.2 is an outlier for the PSO-SVM algorithm. Figure 3.11 represents the accuracy of outliers reported on different datasets. We observe three outliers in the GA-NB algorithm where MC2 and EQ

have lower accuracy values, and PC2 has high accuracy values.

A good SDP model should possess results with high values in the recall, F-measure, and accuracy. Table 3.10 describes the descriptive statistics for the performance measures: recall, F-measure, and accuracy, including the minimum, maximum, mean, median, and standard deviation values, as well as the count of studies from which these values are extracted dataset-wise. It is depicted in Table 3.10 that the majority of the hybrid techniques show mean values of recall in the range of 0.64-0.92. PSO-SVM represented the best mean recall value of 0.97, followed by Firefly-SVM and GFS-AB, with mean recall values of 0.92 and 0.91, respectively. With respect to the median recall values, PSO-SVM scored the best median recall value of 0.99. PSO-ANN scored the worst mean and median recall values of 0.048 and 0.03, which can be attributed to the fact that these recall values are obtained from only one study. With respect to the F-measure values, the hybrid algorithms show mean values in the range of 0.65-0.88, where the maximum mean value is attained by Firefly-SVM with a value of 0.88, followed by the next best PSO-SVM with the mean F-measure value of 0.87. Again, PSO-SVM obtains the highest median F-measure value of 0.92. With respect to the accuracy values, GFS-AB scores the best mean and median accuracy values of 97.57 and 97.9, followed by the next best GFS-LB with mean and median accuracy values of 97.23 and 97.89. Also, the mean accuracy values of a majority of the hybrid algorithms range from 82.58 to 97.57, which is quite acceptable. The statistics in Table 3.10 show the effectiveness of using hybridized techniques in SDP models.

Table 3.10: Descriptive statistics of performance measures for hybrid-based SDP models

Algorithm	Count	Performance measure	Minimum	Maximum	Mean	Median	SD
HIDER	7	Recall	0.14	1	0.66	0.74	0.35
	7	F-measure	0.22	0.94	0.65	0.70	0.25
	3	Accuracy	88.1	97.38	94.14	96.93	5.23
GA-ANN	14	Recall	0.2	1	0.64	0.72	0.25
	6	F-measure	0.41	0.94	0.73	0.72	0.22
	18	Accuracy	76.72	100	89.5	88.07	7.65
GA-DT	10	Recall	0.14	1	0.65	0.74	0.27
	6	F-measure	0.5	0.94	0.74	0.72	0.16
	3	Accuracy	91.68	93.36	92.37	92.09	0.83
PSO-ANN	6	Recall	0	0.1944	0.048	0.03	0.073
	13	Accuracy	80.27	99.99	89.41	87.55	6.03
Firefly-SVM	4	Recall	0.807	0.993	0.92	0.94	0.08
	4	F-measure	0.72	0.99	0.88	0.91	0.12
	4	Accuracy	80.7	99.3	91.9	93.82	8.08
GFS-AB	6	Recall	0.738	1	0.91	0.98	0.1
	3	Accuracy	96.9	97.9	97.57	97.9	0.58
GA-KNN	4	Recall	0.8	0.97	0.87	0.86	0.074
	5	Accuracy	64.52	96.81	84.99	87.6	13.38
GA-SVM	30	Recall	0.3	1	0.84	0.967	0.227
	30	F-measure	0.46	0.997	0.818	0.876	0.166
	35	Accuracy	39	99.77	78.23	76.6	16.62
GA-Fuzzy Logic	3	Recall	0.48	0.78	0.65	0.68	0.15
	3	F-measure	0.6	0.75	0.69	0.73	0.15
	3	Accuracy	68	74.17	72.02	73.91	3.49
PSO-SVM	20	Recall	0.859	1	0.97	0.99	0.045
	24	F-measure	0.63	0.99	0.87	0.92	0.12
	25	Accuracy	58.75	99.59	82.58	85.87	11.29
GFS-LB	3	Accuracy	94.9	97.9	96.07	97.9	2.08
Firefly-NB	8	Accuracy	72.8	88	81.95	83.67	5.87
GA-NB	3	Recall	0.81	0.86	0.84	0.86	0.028
	6	Accuracy	81.44	86.8	84.55	84.93	2.025

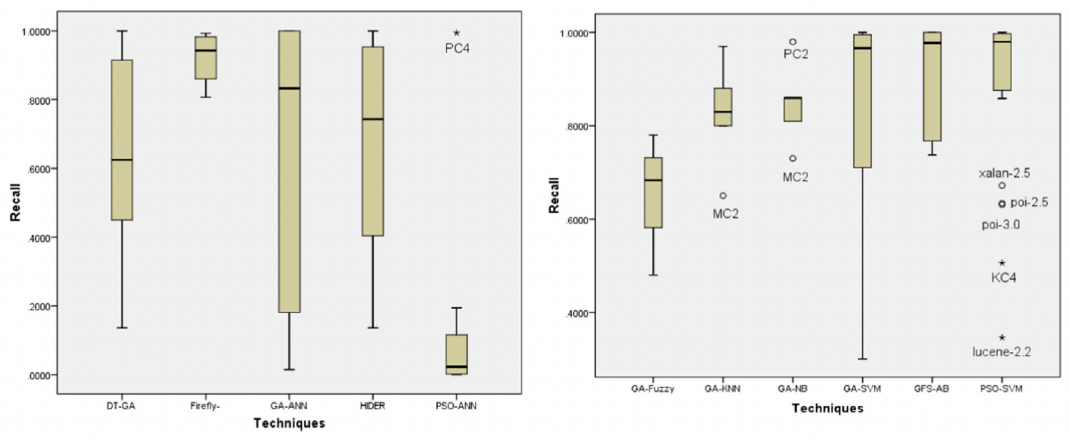

Figure 3.9: Dataset-wise recall outliers for hybridized techniques

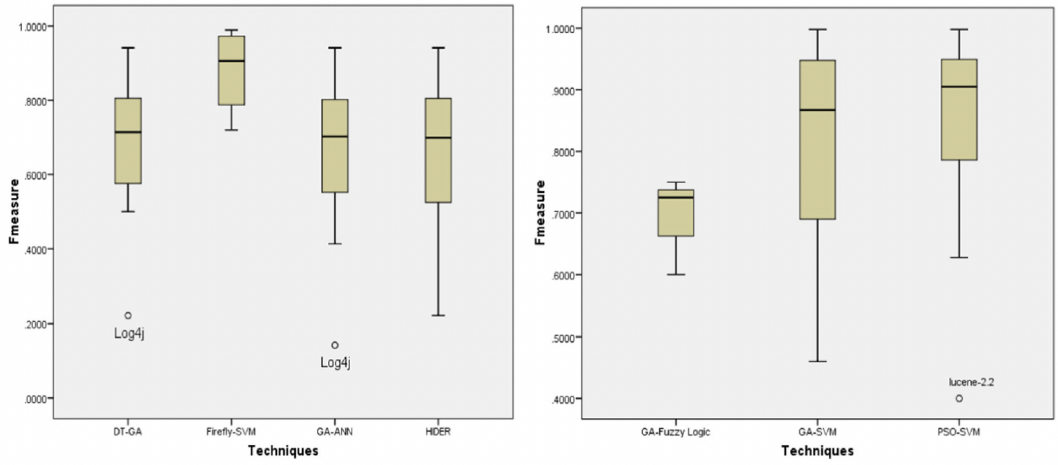

Figure 3.10: Dataset-wise F-measure outliers for hybridized techniques

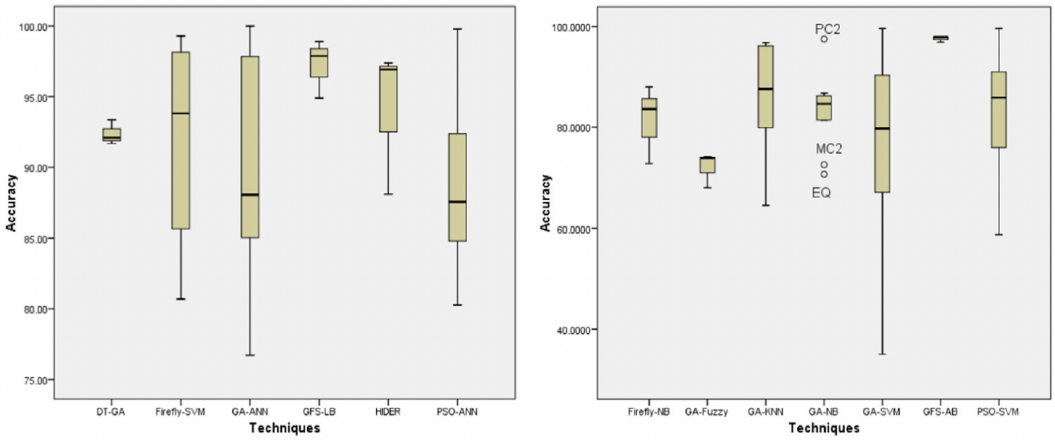

Figure 3.11: Dataset-wise accuracy outliers for hybridized techniques

RQ6.2: What is the comparative performance of hybridized techniques with traditional ML and statistical techniques?

This section investigates the comparative performance of hybridized techniques with statistical and ML techniques used in the literature for the development of SDP models. We analyzed the performance of 13 hybrid algorithms, namely, HIDER, GFS-LB, GA-ANN, GA-DT, PSO-ANN, Firefly-SVM, GFS-AB, GA-KNN, PSO-SVM, GA-SVM, GA-Fuzzy Logic, Firefly-NB, and GA-NB among each other and seven ML techniques, namely Support Vector Machines (SVM), Random Forest (RF), K Nearest Neighbor (KNN), J48, Artificial Neural Network (ANN), Naïve Bayes (NB), C4.5, and two statistical techniques Linear Discriminant Analysis (LDA) and Logistic Regression (LR). These ML/statistical techniques were selected for comparison on the basis that we could extract enough data from at least two primary studies and evaluate them on at least three datasets. We followed the two rules for analyzing the results while comparing the performance of hybridized techniques.

- If any study builds SDP models on the same dataset multiple times, but with different parameter settings, the best performance measure value is chosen for that technique.
- In the case of multiple studies, building the model using the same technique and dataset,

the best performance measure values of all the studies are chosen for that technique.

For statistical validation of comparative performance between several hybrid, ML, and statistical techniques, we used the Wilcoxon signed-rank test with a significance level of 0.05. Tables 3.11, 3.12, and 3.13 show the results for recall, F-measure, and accuracy values, respectively. We use different symbols to represent if the algorithm's performance in the row is superior, inferior, or equivalent to the performance of another algorithm in the column.

- If performance measure value is significantly superior – (↑ *)
- If performance measure value is significantly inferior – (↓ *)
- If performance measure value is superior but is not significant – (↑)
- If performance measure value is inferior but is not significant – (↓)
- If an equal number of positive and negative ranks are obtained – (=)
- If there is not sufficient data to compare between two techniques – (#)

As observed in Table 3.11, GA-SVM outperforms all other algorithms in terms of recall, with significant superior values over PSO-ANN, RF, KNN, NB, J48, ANN, LR, and C4.5 algorithms and non-significantly superior values over other algorithms. PSO-SVM is considered equivalent to GA-SVM and performs better than most algorithms except for GA-DT and GA-Fuzzy Logic. GA-SVM, GFS-AB, and GA-Fuzzy show superior performance over the LR technique. Table 3.12 again indicates that GA-SVM depicted better F-measure results than other algorithms. With respect to accuracy values, GA-SVM gave significantly better results than PSO-SVM, LR, and all ML algorithms, as shown in Table 3.13. For all other algorithms, it gave better accuracy values but was not significant except for HIDER, GFS-LB, GA-DT, and GFS-AB. Five hybrid algorithms, GA-ANN, PSO-SVM, GA-SVM, GA-Fuzzy, and Firefly-NB, performed better than the LR technique in terms of accuracy. Among all

the ML algorithms, SVM gave better performance than five hybrid algorithms (PSO-ANN, Firefly-SVM, GA-KNN, GA-Fuzzy, GA-NB) in terms of accuracy and recall values. In Table 3.12 and Table 3.13, we can observe that GA-SVM gave the best performance measure values out of all the algorithms and hence can be considered the best hybrid algorithm for developing SDP models. Overall, all the hybrid algorithms show superior results over LR and ML techniques. It proves the effectiveness of the hybridized techniques compared to statistical and ML techniques. However, there is a need to collect more studies due to the unavailability of sufficient data for all the performance measures, a more comprehensive comparison is required between the various algorithms.

Table 3.11: Wilcoxon test results for recall values

	HIDER	GA-ANN	GA-DT	PSO-ANN	Firefly-SVM	GFS-AB	GA-KNN	PSO-SVM	GA-SVM	GA-Fuzzy	GA-NB	SVM	RF	KNN	NB	J48	ANN	LR	C4.5	LDA
HIDER	-	(↑)	(↑)	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#	#
GA-ANN	(↓)	-	(↑)	(↑*)	#	#	#	(↓)	(↓)	#	#	(↓)	#	#	#	#	#	#	#	#
GA-DT	(↑)	(↓)	-	#	#	#	#	(↑)	(↓)	#	#	(↓)	#	(↑)	(↑)	#	#	#	#	#
PSO-ANN	#	(↓*)	#	-	#	#	#	(↓*)	(↓*)	#	#	(↓*)	#	#	#	#	#	#	#	#
Firefly-SVM	#	#	#	#	-	#	#	(↓)	(↓)	#	#	(↓)	=	#	#	#	#	#	#	#
GFS-AB	#	#	#	#	#	-	#	=	=	#	#	(↓)	#	#	(↑)	#	(↑)	(↑)	(↑)	(↑)
GA-KNN	#	#	#	#	#	#	-	(↓)	(↓)	#	(↓)	(↓)	#	(↑)	(↑)	=	#	#	#	#
PSO-SVM	#	(↑)	(↓)	(↑*)	(↑)	=	(↑)	-	=	(↓)	(↑)	(↑*)	(↑)	(↑*)	(↑*)	(↑*)	#	#	#	#
GA-SVM	#	(↑)	(↑)	(↑*)	(↑)	=	(↑)	=	-	(↑)	(↑)	(↑)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑)
GA-Fuzzy	#	#	#	#	#	#	#	(↑)	(↓)	-	#	(↓)	#	#	(↓)	#	#	(↑)	#	#
GA-NB	#	#	#	#	#	#	(↑)	(↓)	(↓)	#	-	(↓*)	#	(↑)	(↑*)	(↑)	#	#	#	#

Table 3.12: Wilcoxon test results for F-measure values

	HIDER	GA-ANN	GA-DT	Firefly-SVM	PSO-SVM	GA-SVM	GA-Fuzzy	SVM	RF	KNN	NB	ANN	LR	C4.5	LDA
HIDER	-	(↑)	(↓)	#	#	#	#	#	#	#	#	#	#	#	#
GA-ANN	(↓)	-	(↓)	#	#	#	#	#	#	#	#	#	#	#	#
GA-DT	(↑)	(↑)	-	#	#	#	#	#	#	#	#	#	#	#	#
Firefly-SVM	#	#	#	-	(↓)	(↓)	#	(↓)	(↓)	(↓)	(↓)	#	#	#	#
PSO-SVM	#	#	#	(↑)	-	(↓*)	(↑)	(↑*)	(↑)	#	(↑*)	(↑)	#	(↑)	(↑)
GA-SVM	#	#	#	(↑)	(↑*)	-	(↑)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑)
GA-Fuzzy	#	#	#	#	(↓)	(↓)	-	(↓)	#	#	(↑)	(↓)	(↑)	#	#

Table 3.13: Wilcoxon test results for accuracy values

	HIDER	GFS-LB	GA-ANN	GA-DT	PSO-ANN	Firefly-SVM	GFS-AB	GA-KNN	PSO-SVM	GA-SVM	GA-Fuzzy
HIDER	-	(↓)	(↑)	#	#	#	(↓)	#	(↑)	(↑)	#
GFS-LB	(↑)	-	(↑)	#	#	#	(↑)	#	(↑)	(↑)	#
GA-ANN	(↓)	(↓)	-	(↑)	=	(↓)	(↓)	(↑)	(↑	(↓)	#
GA-DT	#	#	(↓)	-	(↑)	#	#	#	(↑)	(↑)	#
PSO-ANN	#	#	=	(↓)	-	#	#	#	(↓)	(↓)	#
Firefly-SVM	#	#	(↑)	#	#	-	#	#	(↓)	(↓)	#
GFS-AB	(↑)	(↓)	(↑)	#	#	#	-	#	(↑)	(↑)	#
GA-KNN	#	#	(↓)	#	#	#	#	-	(↓)	(↓)	#
PSO-SVM	(↓)	(↓)	(↓)	(↓)	(↑)	(↑)	(↓)	(↑)	-	(↓*)	(↑)
GA-SVM	(↓)	(↓)	(↑)	(↓)	(↑)	(↑)	(↓)	(↑)	(↑*)	-	(↑)
GA-Fuzzy	#	#	#	#	#	#	#	#	(↓)	(↓)	-
Firefly-NB	#	#	#	#	#	#	#	#	(↓)	(↓)	#
GA-NB	#	#	#	#	#	#	#	(↓)	(↑)	(↑)	#

	Firefly-NB	GA-NB	SVM	RF	KNN	NB	J48	ANN	LR	C4.5	LDA
HIDER	#	#	(↑)	#	#	(↑)	#	#	#	(↑)	#
GFS-LB	#	#	(↑)	#	#	(↑)	#	#	#	(↑)	#
GA-ANN	#	#	=	(↑*)	(↑)	(↑*)	ˆ	=	(↑)	(↑)	(↑)
GA-DT	#	#	(↑)	(↑)	#	#	#	#	#	#	#
PSO-ANN	#	#	(↓)	(↑*)	(↑)	(↑)	#	(↑)	#	(↑)	#
Firefly-SVM	#	#	(↓)	=	#	#	#	#	#	#	#
GFS-AB	#	#	(↑)	#	#	(↑)	#	#	#	(↑)	#
GA-KNN	#	(↑)	(↓)	(↑)	(↑)	(↑)	(↑)	(↓)	#	#	#
PSO-SVM	(↑)	(↑)	(↑*)	(↑*)	(↑*)	(↑)	(↑*)	(↑)	(↑*)	(↑*)	(↑)
GA-SVM	(↑)	(↑)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑*)	(↑)
GA-Fuzzy	#	#	(↓)	#	#	(↓)	#	(↓)	(↑)	#	#
Firefly-NB	-	=	(↓)	(↑)	(↓)	=	(↑)	(↓)	(↑)	(↑)	#
GA-NB	=	-	(↓)	#	(↓)	(↑)	(↑)	(↓)	#	#	#

3.4.7 Results specific to RQ7

RQ7: What are the different statistical tests used to study hybridized techniques in SDP?

Statistical tests are essential to obtain quantifiable and statistically verifiable results regarding the predictions produced by a predictive model. These tests provide adequate support to validate the comparisons between different hybridized techniques for building defect prediction models. Out of 72 primary studies, only 41 studies (around 57%) use statistical tests to give evidence of their conclusive results. Parametric and non-parametric tests are the two types of statistical tests used.

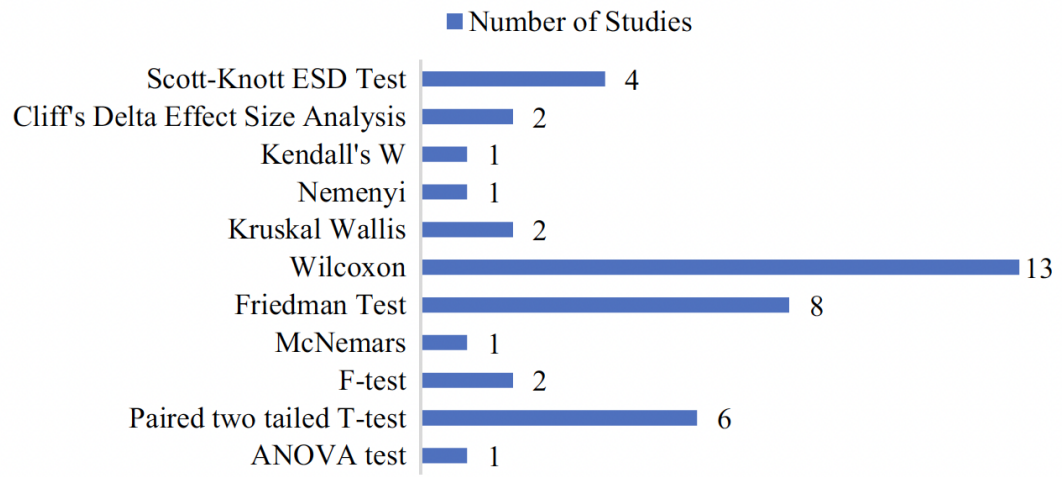

Figure 3.12: Distribution of studies according to statistical analysis

Figure 3.12 describes the distribution of studies based on statistical tests. Non-parametric tests are used in 78% of studies that include statistical analysis, while parametric tests are used in 22%. The wide use of non-parametric tests is supported because they don't require any assumptions about data normality. They are simple to understand and easy to use. Whereas parametric tests require complete knowledge of population distribution, data normality should be true before applying these tests. Some studies have performed parametric tests such as F-

test, Paired two-tailed t-test, and ANOVA tests. Some studies have performed non-parametric tests, such as Kendall's W, Nemenyi, Kruskal Wallis, Friedman, Wilcoxon, McNemar's, Scott-Knott ESD, and Cliff's Delta Effect Size analysis tests. The Wilcoxon test is the most popular of all the studies, which can be used individually for the pairwise comparison of different techniques. It is also generally applied as a post-hoc analysis test after the Friedman test [115]. All the statistical tests used by various primary studies are listed in Table 3.14.

Table 3.14: Statistical tests used by primary studies

Statistical test	**Type**	**Primary studies**
ANOVA	Parametric	HS2
F-test	Parametric	HS55, HS65
Paired two-tailed T-test	Parametric	HS6, HS7, HS11, HS23, HS61, HS66
McNemar's	Non-Parametric	HS4
Friedman	Non-Parametric	HS13, HS28, HS39, HS36, HS40, HS45, HS51, HS56
Wilcoxon	Non-Parametric	HS16, HS17, HS19, HS20, HS28, HS34, HS45, HS43, HS54, HS58, HS65, HS68, HS71
Kruskal-Wallis	Non-Parametric	HS18, HS28
Nemenyi	Non-Parametric	HS28
Kendall's W	Non-Parametric	HS39
Scott-Knott ESD	Non-Parametric	HS68, HS69, HS62, HS63
Cliff's Delta Effect Size Analysis	Non-Parametric	HS68, HS69

3.4.8 Results specific to RQ8

RQ8: What are the strengths and weaknesses of hybridized techniques for SDP?

The strengths and weaknesses of various hybridized techniques, as stated in the primary studies, are highlighted in this section. Hybridized techniques are formed by combining different ML/statistical and search-based techniques for developing SDP models. Few authors have tried replicating the techniques on other datasets like GA-ANN, GA-DT, Firefly-SVM, PSO-SVM, etc. The focus is primarily on creating new hybrid-based models to find interesting results and compare them with the previous works. We present the strengths and weaknesses

of these techniques as supported by studies in Table 3.15. It is observed that the characteristics of various hybridized techniques may only be valid within the study's context. As a result, it is hard to determine whether the application of a particular hybrid technique fits a specific domain. However, to avoid study bias, we only report those characteristics supported by a minimum of two studies.

Table 3.15: Strengths and weaknesses of hybridized techniques for SDP

Study#	Technique	Strengths	Weaknesses
HS36, HS56	GFS-LB	(1) Multiple weak classifiers combined to form a strong classifier resulting in higher performance (2) Learning rate is fast	It requires high interaction between the rules, resulting in a poor-quality rule base
HS36, HS40, HS56	GFS-AB	(1) Multiple weak classifiers combined to form a strong classifier resulting in higher performance (2) Learning rate is fast	It requires high interaction between the rules resulting in a poor-quality rule base
HS10, HS29, HS32, HS39, HS41	GA-ANN	(1) GA has a parallel search method and global optimization properties that enable the ANN to have a higher prediction accuracy and faster convergence (2) GA has interesting searching patterns that select the best weight for a neural network	(1) Crossover and mutation operators are inherently complex in GA (2) Computational cost is increased
HS39, HS42	DT-GA	It selects features better than other greedy algorithms	Suffers from pre-mature convergence problem of local optima when applied on huge datasets and complex problems
HS10, HS41	PSO-ANN	(1) PSO is known to have a strong ability in training ANN (2) No special requirement for encoding and decoding operators for training the ANN (3) Requires very few parameters and produces a robust model	–
HS30, HS33, HS59	Firefly-SVM	Computationally faster and more efficient than GA or PSO	–
HS8, HS21, HS54	PSO-SVM	The model is robust as it is not affected by individual particle error	–
HS12, HS24	GA-Fuzzy Logic	It does not return to the local minima	–

3.4.9 Results specific to RQ9

RQ9: What are the threats to validity in studies using hybridized techniques for SDP?

This section describes all the threats analyzed in the primary studies using the hybridized techniques for SDP. A researcher must examine all the potential threats at the early stages of the experiment to ensure that the results are valid and reliable. It would ensure that proper experiments are conducted so that majority of the threats can be reduced. Additionally, the authors should mention potential threats to inform readers about the study's limitations. We extracted the validity threats from the study sections titled 'Limitations' or 'Threats to validity'. The threats described in the primary studies are classified into internal, external, construct, and conclusion threats. Table 3.16 shows the different categories of threats along with its brief description with respect to study identifiers.

Table 3.16: Threats to validity reported by primary studies

Study#	**Threats to Validity**	**Threat Description**
HS29	Internal Threat	Obtained results don't take into account the severity of the faults
HS36, HS39, HS56	Internal Threat	The size of the software can affect the independent variables, and hence results can be affected
HS27, HS29	Internal Threat	Confounding effect of independent variables and influence of other human factors - developer's experience, familiarity with code
HS2, HS16, HS17, HS18, HS20, HS51, HS57, HS69	Internal Threat	Experimental setting errors and errors in model selection
HS5, HS28, HS36, HS39, HS40, HS56	External Threat	Obtained results are validated on datasets written in a specific language and cannot be generalized

Study#	Threats to Validity	Threat Description
HS2, HS16, HS17, HS19, HS27, HS28, HS29, HS51, HS57, HS58, HS62, HS70	External Threat	Obtained results are validated on specific datasets, which leads to generalizability issues
HS17, HS20, HS27, HS29	External Threat	Obtained results validated based on specific independent variables used
HS2, HS18, HS27, HS51, HS58, HS70	External Threat	Obtained results are validated based on specific algorithms/techniques used
HS16, HS27	External Threat	Obtained results cannot be replicated
HS19, HS20	External Threat	Obtained results are not validated on proprietary datasets
HS2, HS5, HS16, HS18, HS19, HS20, HS28, HS36, HS39, HS40, HS62	Construct Validity	Independent and dependent variables may not be correctly collected
HS2, HS18	Construct Validity	Outliers and noise may not be removed from the training dataset
HS17, HS18, HS20, HS51	Construct Validity	Misinterpretation of concepts and measures
HS16, HS17, HS19, HS20, HS28, HS51	Conclusion Validity	Appropriate statistical tests and significance levels may not be chosen correctly
HS16, HS19, HS28	Conclusion Validity	Performance measures may not be appropriately chosen

28% of the primary studies (HS2, HS5, HS16, HS17, HS18, HS19, HS20, HS27, HS28, HS29, HS36, HS39, HS40, HS51, HS56, HS57, HS58, HS62, HS69, HS70) have mentioned the threats in their studies. Only a few studies (HS5, HS17, HS20, HS27, HS39, HS40, HS51) have discussed some threat mitigation strategies. HS5, HS17, and HS20 reduced the randomness of the algorithms GA and HS by executing them 30 times. HS51 used pair programming to cross-check all the programming and experimental errors. HS27 generalized the results by selecting balanced and unbalanced datasets of different sizes. HS39 also eliminated this threat of generalizability by experimenting with varying distributions of defective classes in datasets.

HS40 mitigated the external threat by using the hold-out validation method.

3.5 Discussion

This review studies and examines the SDP models predicted using hybridized techniques from various viewpoints, like the techniques used in the studies, datasets and metrics used, fitness functions employed, performance metrics, validation methods, and statistical tests for validation. This review also helps determine the pre-processing feature selection/ parameter optimization techniques to develop these predictive models. We analyze the effectiveness of hybridized techniques and show how they are better in prediction capability than statistical and ML techniques using statistical tests. We investigated a total of 72 primary studies after successful refinement based on the quality assessment criteria and inclusion/exclusion factors. The studies are published between January 2000 - December 2021. This process included snowballing, where we scanned the references of the selected papers, and as a result, more relevant studies were found and passed the quality criteria. The information gained from this rigorous systematic review will help the researchers and industrial community to carefully select the hybridized techniques in proposing new SDP models. The significant findings as a result of this systematic review are stated below-

- RQ1 - Hybrid techniques are formed by merging the search-based and ML techniques or search-based and statistical techniques. Search-based techniques are classified into six categories: Local Search, Swarm Intelligence, Nature-Inspired, Bio-inspired, and Evolutionary algorithms. ML techniques are classified into five major categories- Decision Trees, Ensemble Learners, Support Vector Machines, Neural Networks, and other techniques. Statistical techniques such as LR and LDA are also combined with search-based techniques to form hybrid techniques. The most frequently used category in search-based and ML techniques are the evolutionary techniques (54% of the studies)

and the SVM technique (33% of the studies) used to form hybrid-based models.

- RQ2 - The primary studies included in this systematic review are published between 2000 - 2021. 92% of the primary studies are published between 2013 and 2021. After 2016, we noticed a rise in the number of studies as a result of the application of deep learning techniques such as CNN, LRNN, DNN, ELM, and KELM.

- RQ3 - This systematic review includes 72 papers, with 72% of the primary studies (51) published in journals, 23% of the studies (16) published in conferences, and 7% of the studies (5) published as book chapters.

- RQ4 - SDP models using hybridized techniques are developed on various datasets- NASA, Promise, Open-source, academic and industrial datasets. Most studies use traditional or static metrics (65%) as the predictor variables. Composite metrics that include static and OO metrics are used in 33% of the studies, OO metrics alone are used in 3% of the studies, and 17% use miscellaneous metrics like change metrics, process, and product metrics, etc.

  For data pre-processing, 46% of the studies use search-based techniques for selecting features, and 33% use search-based techniques for parameter optimization. F-measure, Accuracy, and Recall are the widely used performance metrics in primary studies for evaluating SDP models. K-Fold validation is employed in most studies, whereas hold-out validation is used in some studies where the dataset size is huge.

- RQ5 - The most frequently used fitness functions are accuracy and its variants and the multi-objective function of maximizing the recall and minimizing the false positive rate. It becomes crucial for researchers to choose the correct fitness function to achieve the best results because the fitness function impacts the efficiency of the overall model prediction.

- RQ6 - The predictive capability of hybridized techniques in terms of recall ranged between mean values of 0.64 and 0.97, mean F-measure values ranged from 0.65 - 0.88, and mean accuracy values ranged from 72.02 - 97.57. The Wilcoxon sign test results show that GA-SVM outperforms the majority of other hybrid approaches and ML and statistical techniques with respect to all three performance measures. GA is the most widely used evolutionary algorithm, and PSO is more commonly used in the category of swarm intelligence. Overall, the performance of hybrid techniques is superior in comparison to ML and statistical-based models.

- RQ7 - 57% of the primary studies use statistical tests. Friedman and Wilcoxon's tests are the most commonly used non-parametric tests, as they don't require any assumptions about data distribution.

- RQ8 - Only a few papers have looked into the strengths and weaknesses of the hybridized techniques used to develop SDP models. Generally, a hybrid technique may have specific properties suitable for a particular predictive modeling task, which can affect the predictive capability of a model. Therefore, it is necessary to thoroughly examine a technique before using it in any predictive model.

- RQ9 - Only 33% of the studies have mentioned threats to validity in their research works. We classified these threats into internal, external, construct, and conclusion. It is observed that there are more generalizability issues in terms of the specific datasets used, programming language, techniques used, etc., in all the experiments conducted by the studies. Very few studies mitigated these threats to yield valid and reliable results.

Chapter 4

Software Defect Prediction based on Binary Particle Swarm Optimization and ANN

4.1 Introduction

Software testing is an indispensable step in the software development life cycle process for multiple reasons, such as increasing software size, software complexity, and continuous updates in the system. Once the software development lifecycle begins, detecting and fixing defects becomes the most expensive process. Correct prediction of software defect-prone modules would enhance the testing process by concentrating primarily on defect-prone modules, detecting candidates for refactoring that are likely to be defect-prone, and improving the software quality [201]. Software metrics play a significant role in defect prediction [202]. These are the internal attributes that help in estimating the number of defects in software. Previous research findings show that the

software metrics are essential and help categorize software modules into two classes-defective and non-defective [203]. Often, many of these metrics in a dataset may not help predict defects. It may be due to insignificant and redundant input(independent) variables in the dataset, which can result in deteriorated performance on unseen data and overfitting problems. This issue is referred to as the curse of dimensionality [204]. The feature selection technique is used to address this issue. It is an essential preprocessing phase in which we select and exclude irrelevant features to create a predictive model. Besides high dimensionality, one more issue, an imbalance in the dataset, affects the performance of ML algorithms. It is a problem wherein, in a dataset, one class is in the minority (very rare), whereas another class is in the majority [113]. SMOTE is a well-known balancing technique where synthetic samples from the minority class are added to the dataset to balance the class distribution.

To deal with this problem of feature selection in SDP, search-based techniques for optimization problems have proved their efficiency in SDP [18, 54, 143, 146, 204, 205]. PSO [85] is one such evolutionary algorithm whose behaviour is inspired by a swarm of birds in a population continuously moving in search space and updating their positions until a stopping condition or an optimal solution is reached. This research aims to use PSO as a feature selection algorithm as it addresses the problem of high dimensionality and requires less memory and runtime while being computationally inexpensive [206]. ANN algorithm is used for classification as it has the potential to handle classification and regression problems very effectively in the SDP domain [18, 207]. Among all the ML algorithms, ANN provides superior performance to linear models and is known for its non-linear properties. To improve the prediction performance of real-world software projects, we offer an effective hybrid defect prediction model employing a wrapper technique integrated with SMOTE in this research study. The following are the main contributions of this chapter:

- Binary Particle Swarm Optimization utilizing the AUC as a fitness function.
- SMOTE for re-balancing the imbalanced datasets.
- ANN is used as a classification algorithm for predicting software defects.
- Statistical validation using the Friedman and Wilcoxon Test with AUC and G-mean as performance measures.

The research questions have been formulated as follows:

- RQ1: What is the prediction capability of the hybridized BPSO-SMOTE-ANN model for SDP? Does the performance of BPSO-SMOTE-ANN improve after applying the balancing technique?
- RQ2: What is the performance of BPSO-SMOTE-ANN in comparison with other ML techniques?
- RQ3: Whether BPSO-SMOTE-ANN is the best predictive technique for SDP?

The structure of this chapter is outlined as follows. The BPSO-based Feature selection technique is discussed in Section 4.2.1. The proposed methodology is described in Section 4.2.4. The results and analysis part of the chapter that answers the research questions is discussed in Section 4.3. Section 4.4 provides the discussion of this chapter. The results of this chapter are presented in the conference paper [208].

4.2 Research Methodology

This section encompasses the key steps of the empirical study, including the proposed feature selection technique, datasets, variable selection, data preprocessing, performance evaluation measures, model validation, statistical tests, and comparison with baseline techniques.

4.2.1 Proposed BPSO-based Feature Selection

We have used the binary variant of the PSO algorithm as a wrapper-based feature selection method to develop an efficient SDP model for reducing the dimensionality of the input search space. In this feature selection method, we have used the fitness function to maximize the classification performance (AUC) value. The details of this feature selection method can be found in subsection 2.6.5 of Chapter 2.

4.2.2 Datasets and Variables

We have selected five publicly accessible datasets from the NASA software project repository, namely CM1, KC1, PC3, PC4, and MW1. The detailed description of these datasets can be found in Section 2.7.2. These datasets are widely used in the field of SDP and consist exclusively of numeric attributes. The NASA datasets were selected considering different project sizes and programming languages. Each project contains over 400 instances with varying defect ratios, making them well-suited for studies based on static code metrics.

4.2.3 Data Preprocessing

It is very important to normalise the input dataset as the first step of data preprocessing because every dimension in an input space may contain a different range of values. In ANN training, we usually normalize the data so that all input values range between [0, 1]. It improves the training process of the ANN and overcomes the model learning problem. In this chapter, we have used the min-max normalization [18].

4.2.4 Model Development and Validation

We devise an SDP model that can help efficiently identify modules into defective and non-defective classes using a hybrid approach of ANN and BPSO with the SMOTE imbalance technique. Figure 4.1 explains a step-by-step procedure followed in our research for model prediction. Algorithm 4.2 describes the pseudocode for the proposed methodology. After the predictive model is developed, we analyze its performance with metrics like AUC and G-mean. This proposed hybrid model is compared to other ML algorithms and one evolutionary algorithm using the Friedman and Wilcoxon tests.

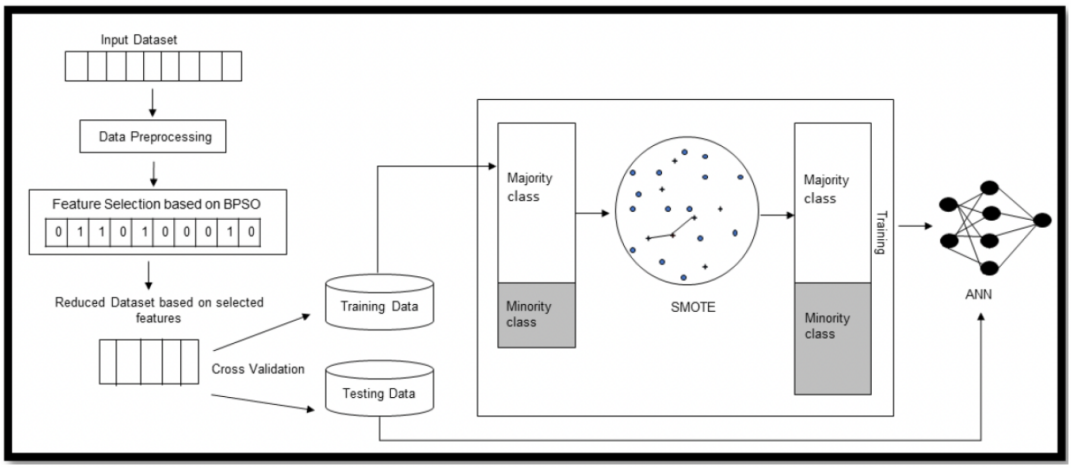

Figure 4.1: Proposed methodology

We use a ten-fold cross-validation method, where the value of $k = 10$. It implies that the input dataset is split into 10 portions, where nine portions of the data are used for training purposes, one portion is used for testing purposes, and the process repeats 10 times. The training data is used for model development, and the testing data is used for making predictions. Training and testing data are randomly selected in each iteration [209].

Algorithm 1 Pseudocode for BPSO-SMOTE-ANN

Initialization of particles with position and velocity values in the range [0, 1].
For maximum no. of iterations:
For each particle in the population:
Calculate the fitness value for each particle.
Update the particle's best position if the particle's fitness value is greater than the particle's best fitness.
Update the global best position if the particle's fitness value is greater than the global best fitness value.
Update the particle's position and velocity.
Return global best as a feature subset.
Divide the dataset into training and test datasets by applying 10-fold cross-validation.
For the training dataset obtained in the previous step, apply SMOTE to oversample the minority class samples.
Build an ANN model on the oversampled training data.
Make predictions on the testing dataset to identify if it is defective or non-defective.
Compute performance measures like AUC, Recall, Precision, and G-mean scores on the test dataset for each fold and return their average scores.

Figure 4.2: Algorithm for BPSO-SMOTE-ANN

4.2.5 Parameter Settings

The parameter settings required for implementing BPSO and ANN are specified in Table 4.1. The parameters were carefully chosen after reviewing the literature and considering commonly adopted configurations in similar studies.

4.2.6 Performance Measures

In our experiments, the ROC-AUC and G-mean were employed as performance metrics to evaluate the proposed models, as these are considered stable metrics than

Table 4.1: Parameters for the BPSO algorithm

Parameters	**Values**
Number of particles	60
Maximum iterations	100
Acceleration constants ($c_1 = c_2$)	2
Inertia weight	0.45
Number of runs	20

Table 4.2: Parameters for the ANN algorithm

Parameters	**Values**
Initial weights for ANN	-1.0 to 1.0
Learning rate	0.1
Number of training epochs	250
Activation function for hidden and output layer	Sigmoid function

traditional measures such as accuracy when handling imbalanced and noisy datasets. A detailed explanation of these metrics is provided in Section 2.7.7.

4.2.7 Statistical Evaluation

To strengthen the findings based on ROC-AUC and G-mean values, statistical validation was conducted. The Friedman test was first applied to assess whether significant performance differences existed among the classifiers. When significant differences were identified, a post hoc Wilcoxon signed-rank test was employed for pairwise comparisons. This approach offered a more rigorous evaluation and deeper insight into classifier performance variations. A detailed discussion of these statistical tests and their application in this study is presented in Section 2.7.8.

4.2.8 Baseline Techniques

The study compares the results obtained from the hybridized BPSO-SMOTE-ANN model with various ML algorithms K-Nearest Neighbor (KNN), Naïve Bayes (NB),

Support Vector Machine (SVM), Decision Trees (CART), and one evolutionary algorithm, namely–Genetic Algorithms (GA). The algorithms are implemented in Python 3.8 on Spyder IDE with an Intel Core i7 processor and RAM of 16 GB. These algorithms have been extensively used in previous literature for predictive modelling in the context of software quality. The details of these techniques have been given in Section 2.6

4.3 Results and Analysis

The following sections explain the results and answers to the research questions.

4.3.1 Results specific to RQ1

RQ1: What is the prediction capability of the hybridized BPSO-SMOTE-ANN model for SDP? Does the performance of BPSO-SMOTE-ANN improve after applying the balancing technique?

We evaluate the effectiveness of the BPSO-SMOTE-ANN model for SDP to answer this research question. Five publicly available datasets were used to test the BPSO-SMOTE-ANN model. Table 4.3 summarizes the original number of features in the datasets, the average feature selection size ratio, and the average number of features selected. Since PSO is stochastic, we perform 20 runs and compute the average values on all five datasets. It is observed that in total, 56.4%, 44.3%, 61.2%, 58.2%, and 71.5% attributes are reduced for CM1, KC1, PC3, PC4, and MW1 datasets, respectively, thereby making an average of 58% attribute reduction which is substantially significant. It means that, on average, less than half of the features are required to obtain the highest classification performance, and this procedure takes much less time than it would take to execute on all sets of features. Table 4.4 reports

the Average AUC, Recall, Precision, and G-mean values for the three predictive models ANN, BPSO-ANN, and BPSO-SMOTE-ANN developed on datasets. The following observations were made from the above table:

1. The average AUC and G-mean values of BPSO-SMOTE-ANN for all five datasets lie in the range of 0.74–0.91 and 0.62–0.86, which are in the acceptable range (G-mean $>$ 0.5 and AUC $>$ 0.7) in all the cases; therefore we designate the BPSO-SMOTE-ANN to be accurate and reliable.

2. The balancing technique increases the AUC, Recall, Precision, and G-mean scores in all the datasets except for the PC3 dataset, where the AUC value decreased, but overall, the G-mean values have improved. It suggests the importance of SMOTE as it enhances the ability to predict minority class instances.

Table 4.3: Comparison of results - with and without feature selection

Dataset	Original No. of Features	Avg. Feature Selection Size	No. of Features Selected after BPSO
CM1	21	0.435714	9.15
KC1	21	0.557143	11.7
PC3	37	0.383784	14.2
PC4	37	0.417568	15.45
MW1	37	0.285135	10.55

The average AUC value is higher as BPSO and SMOTE are applied, reducing the correlation between software features, and indicating the importance of removing redundant features in predicting software defects. In conclusion, we can say that the feature selection and the SMOTE technique have enhanced the efficiency of SDP.

4.3.2 Results specific to RQ2

RQ2: What is the performance of BPSO-SMOTE-ANN in comparison with other ML techniques?

Table 4.4: Comparison of ANN, BPSO-ANN, and BPSO-SMOTE-ANN models in terms of AUC, Recall, Precision, and G-mean

Dataset	Avg. AUC			Avg. Recall		
	ANN	BPSO-ANN	BPSO-SMOTE-ANN	ANN	BPSO-ANN	BPSO-SMOTE-ANN
CM1	0.72	0.7394	0.7409	0	0.001	0.6818
KC1	0.7853	0.7895	0.7929	0.025	0.0409	0.7923
PC3	0.8215	0.8020	0.7959	0	0.0067	0.2329
PC4	0.9153	0.9132	0.9192	0.344	0.2971	0.8818
MW1	0.7177	0.7617	0.7662	0	0.0525	0.1667

Dataset	Avg. Precision			Avg. G-mean		
	ANN	BPSO-ANN	BPSO-SMOTE-ANN	ANN	BPSO-ANN	BPSO-SMOTE-ANN
CM1	0	0.005	0.223	0	0.0023	0.6816
KC1	0.18	0.2166	0.2895	0.0707	0.1086	0.6984
PC3	0	0.0067	0.2329	0	0.0025	0.7285
PC4	0.3524	0.4146	0.4584	0.4357	0.4108	0.8634
MW1	0	0.0315	0.1931	0	0.0319	0.6201

We compared our proposed model with KNN, NB, SVM, CART, and GA, which were previously used in the literature for defect prediction [49, 143, 210]. 10-fold cross-validation is used on all five datasets for model validation. Table 4.5 and Table 4.6 show the average AUC and G-mean scores for the algorithms used for comparison. For the CM1 dataset, the average AUC and G-mean values ranged from 0.54-0.75 and 0-0.6816. GA-SMOTE-ANN obtains the highest score of AUC, and BPSO-SMOTE-ANN obtains the second-highest AUC = 0.74. BPSO-SMOTE-ANN scored the highest with G-mean = 0.68. For the KC1 dataset, the AUC and G-mean values ranged from 0.61-0.80 and 0.07-0.70. GA-SMOTE-ANN and BPSO-SMOTE-ANN obtained the highest AUC = 0.80, and ANN performed the third best with AUC = 0.79. BPSO-SMOTE-ANN scored the highest G-mean = 0.70. For the PC3 dataset, the AUC and G-mean values ranged from 0.63-0.82 and 0-0.73. A simple ANN (Backpropagation model) obtains the highest AUC = 0.82 with no feature selection applied. However, the G-mean value is improved with feature selection and SMOTE, with 0.73 as the highest score. For the PC4 dataset, the AUC and G-mean values

ranged from 0.71-0.93 and 0.34-0.86. GA-SMOTE-ANN obtains the highest AUC score, and BPSO-SMOTE-ANN obtains the second-highest score of 0.92. BPSO-SMOTE-ANN scored the highest G-mean = 0.86. For the MW1 dataset, the average AUC and G-mean values ranged from 0.58–0.77 and 0–0.63. NB scored the highest AUC = 0.77 and the highest G-mean = 0.63. We have compared our studies with some previous literature also. Wahono and Suryana [54, 137] applied PSO as a feature selection technique on ten different classifiers-SVM, Naïve Bayes, Logistic Regression, Linear Discriminant Analysis, k-NN, K*, Neural Network-BP, CART, C4.5, and Random Forest. They also used the same NASA datasets as used in our research. We observed that the AUC values of our proposed approach are the best in comparison with all other classifiers (with PSO + bagging and GA + bagging) for the MW1, PC3, and PC4 datasets. It demonstrates that combining the imbalance technique with feature selection improves the SDP performance. Nagappan et al. [142] have also proved that ANN-PSO outperformed ANN-GA and ANN-Backpropagation in terms of accuracy by 1.82% and 5.94% in object-oriented systems.

Table 4.5: Average AUC scores of proposed hybrid classifier and other techniques

Datasets/Algorithms	**CM1**	**KC1**	**PC3**	**PC4**	**MW1**
DT	0.5418	0.6127	0.6304	0.7104	0.5835
SVM	0.6510	0.6815	0.7217	0.8957	0.7461
NB	0.7327	0.7915	0.7623	0.8462	0.7696
KNN	0.6615	0.7259	0.7400	0.8403	0.7507
ANN	0.7200	0.7853	0.8214	0.9152	0.7177
GA-ANN	0.7102	0.7968	0.7999	0.9101	0.7673
GA-SMOTE-ANN	0.7457	0.7969	0.7889	0.9229	0.7488
BPSO-ANN	0.7394	0.7895	0.8020	0.9132	0.7617
BPSO-SMOTE-ANN	0.7408	0.7928	0.7958	0.9191	0.7661

Table 4.6: Average G-mean scores of proposed hybrid classifier and other techniques

Datasets/Algorithms	CM1	KC1	PC3	PC4	MW1
DT	0.3093	0.5603	0.5470	0.6851	0.2452
SVM	0.0000	0.3959	0.0000	0.3812	0.0000
NB	0.4935	0.5773	0.5841	0.6228	0.6278
KNN	0.0894	0.4908	0.3277	0.5488	0.2188
ANN	0.0000	0.0706	0.0000	0.4356	0.0000
GA-ANN	0.0044	0.0876	0.0000	0.3382	0.0183
GA-SMOTE-ANN	0.6756	0.6977	0.7225	0.8504	0.6122
BPSO-ANN	0.0022	0.1086	0.0025	0.4107	0.0319
BPSO-SMOTE-ANN	0.6816	0.6984	0.7285	0.8634	0.6201

4.3.3 Results specific to RQ3

RQ3: Whether BPSO-SMOTE-ANN is the best predictive technique for software fault prediction?

We used the Friedman test on AUC and G-mean to show whether BPSO-SMOTE-ANN significantly performs better than other techniques. Table 4.7 depicts the mean rank obtained by various techniques on all five datasets.

Table 4.7: Friedman Test results based on AUC and G-mean

Algorithms	Mean ranks (AUC)	Mean ranks (G-mean)
DT	1.00	6.00
SVM	2.80	2.00
NB	6.00	7.00
KNN	3.00	5.40
ANN	6.20	2.00
BPSO-ANN	6.80	3.60
BPSO-SMOTE-ANN	7.00	8.80
GA-ANN	6.00	2.40
GA-SMOTE-ANN	6.20	7.80

BPSO-SMOTE-ANN obtained the highest rank of 7.00 and 8.80 with respect to AUC and G-mean out of all the techniques for determining defect-prone classes. We got the p-value = 0.002 and chi-square value = 24.64 with 8 degrees of freedom for AUC. We got the p-value = 0.000 and chi-square value = 36.471 with 8 degrees of

Table 4.8: Wilcoxon Test results based on AUC and G-mean

Techniques-Pair	**p-values with respect to AUC**	**p-values with respect to G-mean**
BPSO-SMOTE-ANN – DT	0.043 → S	0.043 → S
BPSO-SMOTE-ANN – SVM	0.043 → S	0.043 → S
BPSO-SMOTE-ANN – NB	0.225 → NS	0.080 → NS
BPSO-SMOTE-ANN – KNN	0.043 → S	0.043 → S
BPSO-SMOTE-ANN – ANN	0.893 → NS	0.043 → S
BPSO-SMOTE-ANN – BPSO-ANN	0.893 → NS	0.043 → S
BPSO-SMOTE-ANN – GA-ANN	0.345 → NS	0.043 → S
BPSO-SMOTE-ANN – GA-SMOTE-ANN	0.500 → NS	0.043 → S

freedom for G-mean. The p-value < 0.05 in both cases signifies that the results are substantial and the proposed model performs significantly better than other techniques. We further applied the Wilcoxon Test that performs pairwise comparisons with BPSO-SMOTE-ANN as a control technique. The test was conducted on the AUC and G-mean measures for all five datasets at a cut-off point of 0.05. Table 4.8 depicts the results of the Wilcoxon test, where the **S** symbol denotes a significant difference, and **NS** denotes no significant difference. BPSO-SMOTE-ANN significantly outperforms DT, SVM, and KNN algorithms with respect to AUC values over all the datasets ($p < 0.05$). But there is no statistical difference in the performance between BPSO-SMOTE-ANN and NB, ANN, BPSO-ANN, GA-ANN, and GA-SMOTE-ANN with respect to AUC values. However, the p-values with respect to G-mean suggest that BPSO-SMOTE-ANN outperformed DT, SVM, KNN, BPSO-ANN, GA-ANN, and GA-SMOTE-ANN except for NB. Thus, we can say that the performance of the proposed hybrid classifier was competent with GA as it performs equally well as PSO [] and is better than ML classifiers.

4.4 Discussion

In this chapter, we proposed a hybrid BPSO-SMOTE-ANN model that demonstrates significant improvements in SDP, as evidenced by enhanced AUC and G-mean values across five open-source software systems from the NASA dataset. By integrating BPSO for feature selection, SMOTE for addressing class imbalance, and ANN for classification, the model effectively tackles two major challenges in SDP: high dimensionality and class imbalance. The use of AUC as the fitness function in BPSO ensures that feature selection is directly aligned with maximizing predictive performance, while SMOTE's synthetic sampling balances the minority and majority classes, leading to more robust and generalizable models.

When compared to other machine learning and hybrid approaches reported in the literature, the BPSO-SMOTE-ANN model consistently outperforms or matches the state-of-the-art techniques. For instance, when compared with KNN, SVM, CART, NB, and evolutionary models such as GA-ANN, the BPSO-SMOTE-ANN model demonstrated superior or competitive performance in almost all scenarios. The AUC values (ranging from 0.74 to 0.91) and G-mean values (ranging from 0.62 to 0.86) across the five NASA datasets affirm the reliability and robustness of the hybrid approach. Previous studies have employed combinations such as PCA+ANN, GA+DNN, and various PSO-based hybrids, often reporting improvements in accuracy or AUC but not always addressing both feature redundancy and class imbalance simultaneously.

The Friedman test and Wilcoxon post-hoc test confirm the statistical significance of performance improvements. BPSO-SMOTE-ANN attained the highest ranks (7.00 and 8.80) for AUC and G-mean, respectively. This underscores the effectiveness of BPSO-SMOTE-ANN in identifying the correlation between metrics and defect-proneness.

There is an impact of selecting a few metrics and eliminating irrelevant or redundant features for training and predicting results on the ANN. It decreased the training time and improved the performance of the SDP model by focusing only on defect-prone software modules that ultimately helped minimise the cost of maintenance and would result in higher-quality software.

Chapter 5

Software Defect Prediction based on Multi-Filter Wrapper Feature Selection and Deep Neural Network with Attention mechanism

5.1 Introduction

SDP plays a vital role in software quality assurance by leveraging data mining and ML techniques using the software metrics and historical defect data to build predictive models [211]. Early defect estimation during development helps developers and managers allocate testing resources efficiently, reducing cost and effort. Researchers have shown that certain software metrics strongly correlate with defects; however, redundant and irrelevant features often degrade prediction quality due to the curse of dimensionality [54, 62, 145, 212]. Feature selection methods address this issue by

identifying the most informative subset of attributes while preserving classification accuracy.

Feature selection approaches are generally categorized as filter, wrapper, or hybrid methods. Filter techniques evaluate feature relevance based on data properties, while wrappers employ learning algorithms to search for optimal subsets. Although filters are computationally efficient, wrappers often achieve superior results. Hybrid methods combine their strengths to balance efficiency and predictive accuracy.

Metaheuristic algorithms such as the Whale Optimization Algorithm (WOA) have shown promise in addressing the difficulties presented by high-dimensional datasets. These techniques optimize the feature subset by balancing exploration and exploitation. Despite WOA's superior exploration capability, it may suffer from local optima. To overcome this, opposition-based learning has been integrated into WOA (OBWOA) to enhance solution quality [213].

In this chapter, we propose a multi-filter wrapper feature selection (MFWFS) technique that integrates three filters-Information Gain, chi-square, and Relief-F with wrapper-based OBWOA to eliminate redundant features and select optimal subsets. To further improve predictive capability, we design a unified defect predictor based on 1D-CNN with an attention mechanism that integrates the automatic features generated from the output of CNN and features obtained from the MFWFS technique. Unlike 2D-CNN, which was originally designed for image data, 1D-CNN offers lower complexity, fewer parameters, and has demonstrated strong performance in domains such as ECG monitoring and fault detection [214]. The hybrid model is evaluated on 17 software defect datasets obtained from NASA, AEEEM, and Promise repositories. We have computed the performance metrics-AUC, G-mean, F-measure, and MCC to validate and analyze the classification performance of the proposed model. The predictive capability of the proposed model is compared to other state-of-the-art ML and hybrid algorithms present in the literature to depict that our model has produced better

results across all the datasets. We employed the Wilcoxon test to further establish the statistical significance of our results. This methodology is a noteworthy development in SDP, demonstrating how a hybrid feature selection approach can enhance the predictive capability and effectiveness of software fault prediction.

The research questions guiding this study are:

RQ1. How efficient is the SDP model developed using 1D-CNN with and without multi-filter wrapper feature selection?

The primary objective of RQ1 is to evaluate the effectiveness of incorporating MFWFS into the 1D-CNN model for SDP. Specifically, it investigates whether applying a hybrid feature selection technique improves classification performance compared to using 1D-CNN alone. The analysis measures improvements across AUC, G-mean, F-measure, and MCC values to determine the impact of feature selection on defect prediction models. The results are statistically validated using a Wilcoxon pairwise comparison test.

RQ2. How efficient is the SDP model developed using MFWFS-1D-CNN with and without attention?

The primary objective of RQ2 is to examine whether incorporating an attention mechanism enhances the performance of the MFWFS-1D-CNN model for SDP. It aims to compare the predictive capability of models with and without attention across multiple datasets using F-measure, AUC, G-mean, and MCC values. This helps determine the significance of the attention layer, thus enabling the precise identification of the defects in the software. The results are statistically validated using a Wilcoxon pairwise comparison test.

RQ3. How effective is the SDP model developed using the proposed hybrid model MFWFS-1D-CNN-Attention in comparison with the models created with individual filters and a wrapper-based CNN model?

The primary objective of RQ3 is to assess the effectiveness of the proposed hybrid

model (MFWFS-1D-CNN-Attention) against models developed using individual filter methods (IG, CS, RF) and the wrapper-based OBWOA method. Specifically, it examines whether integrating multiple filters with OBWOA and embedding them within a CNN with attention leads to superior classification performance. The evaluation relies on F-measure, AUC, G-mean, and MCC values to establish the robustness and consistency of the hybrid model across diverse datasets.

RQ4. How effective is the SDP model developed using the proposed hybrid classifier (MFWFS-1D-CNN-Attention) in comparison with other state-of-the-art ML and hybrid techniques?

The primary objective of RQ4 is to evaluate the effectiveness of the proposed hybrid classifier (MFWFS-1D-CNN-Attention) in comparison with other state-of-the-art ML and hybrid techniques reported in the literature. The aim is to determine whether integrating multi-filter wrapper feature selection with a CNN-attention framework achieves superior performance across benchmark datasets.

The remainder of the chapter is organised as follows. Section 5.2 presents the in-depth explanation of the research methodology adopted for implementation. Section 5.3 describes the experimental layout of the paper, giving details of the dataset collection, preprocessing, parameter settings, performance measures and statistical analysis. Section 5.4 presents the obtained results and analyses them in the form of answers to the research questions. Section 5.5 reports the key findings of this chapter. The results of the chapter are published in [215].

5.2 Research Methodology

This research paper presents a hybrid of the multi-filter wrapper feature selection technique with the attention-based 1D-CNN model. The proposed methodology is explained in Fig.6.1.

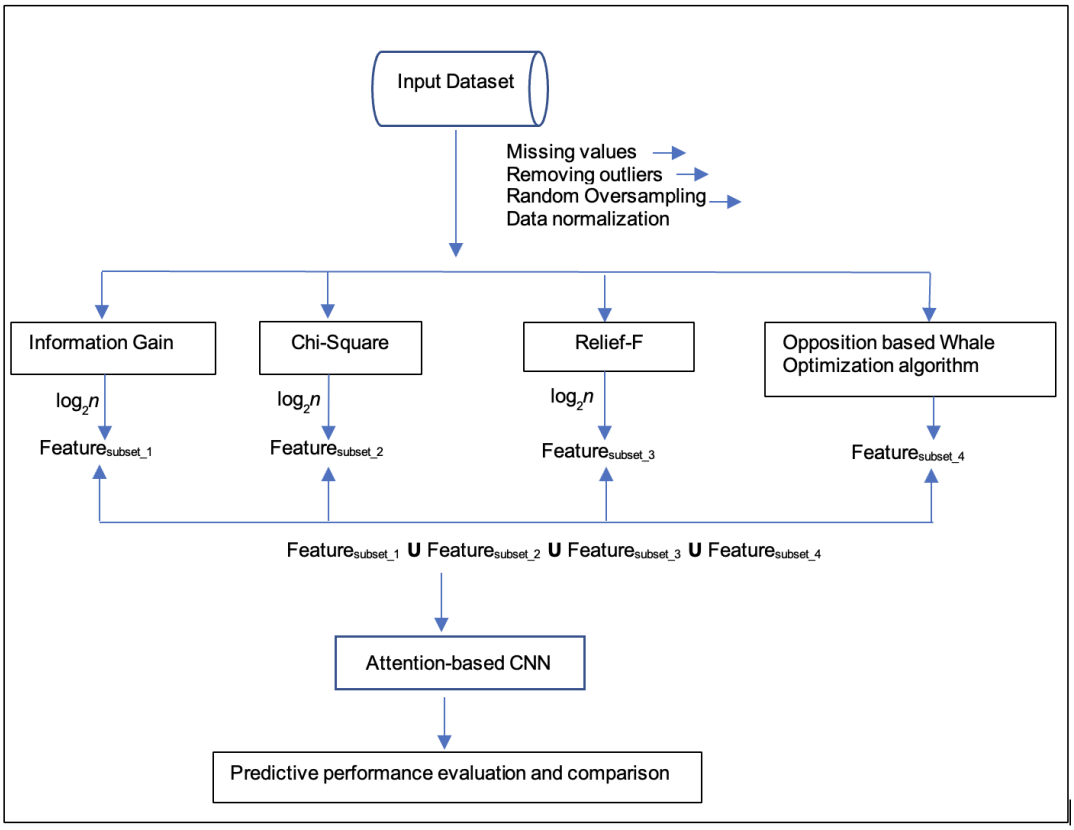

Figure 5.1: Proposed methodology

5.2.1 Feature Selection based on Multi-Filter Wrapper Technique

The proposed MFWFS technique aims to analyze and combine the computational strengths of various independent filter and wrapper-based feature selection methods. The filter methods (IG, CS, and RF) evaluate the features on the basis of their usefulness toward better prediction performance to construct an optimal subset of features with excellent prediction characteristics. The swarm-based oppositional whale optimization algorithm helps accelerate the convergence to the global optimum by finding solutions in opposite directions, increasing the chances of finding better regions effectively [216]. The MFWFS is carried out using the algorithm described

below.

Input: Dataset D

Output: Feature Subset

a. IG_Set = Apply Information Gain FS method

b. CS_Set = Apply Chi-square FS method

c. RF_Set = Apply Relief-F method

d. Filter_Set = $\log_2 n$(IG_Set) $\cup$ $\log_2 n$(e. CS_Set) $\cup$ $\log_2 n$(RF_Set)

f. Wrapper_Set = Apply Opposition-Based Whale Optimization algorithm

g. Reduced_Feature_Set = Filter_Set $\cup$ Wrapper_Set

The input software defect datasets are split into training and test datasets in the ratio of 70:30. To determine the ranks of all the features for each filter method, we independently apply the information gain, chi-square, and relief-F methods to the training set. The top-ranked features are selected from the generated rank list using $\log_2 n$, where n stands for the total number of features in every dataset. Based on previous empirical studies, the choice of $\log_2 n$ is made [108, 217]. As a result, software defect datasets with fewer optimal features will be produced. In OBWOA, each solution refers to a feature subset for every project that is the 1D binary vector with n elements, each of which represents n features and has a value of either '0' or '1'. Two optimization goals should be considered - reducing the number of features and maximizing the classification performance. We combine the above two goals into an objective (fitness) function in Equation 5.1 [55].

$$\text{Fitness} = \alpha \cdot (1 - p) + (1 - \alpha) \cdot \frac{|S|}{|N|} \tag{5.1}$$

where α represents a user-defined hyperparameter $\in [0, 1]$ that strikes a balance between both objectives. p denotes AUC measure, $|S|$ represents feature subset length, and $|N|$ indicates total number of features. $\alpha = 0.99$ as suggested in the literature [88, 89]. We apply the OBWOA on the original training input dataset to extract the subset of features with high AUC values, using KNN ($K = 5$) as the classifier. To provide the best possible set of features, feature subsets from filter and wrapper approaches are combined. The MFWFS is used only for feature selection for every separate software project. For classification purposes, we use the attention-based 1D-CNN defect predictor.

The primary contribution of the above feature selection technique is to explore the complementary nature of filter with wrapper feature selection, with the aim to reduce the feature subset of software defect datasets and enhance the classification performance without compromising on the computational cost. The selection process of features for filter methods is fast, as there is no learning algorithm required, whereas in the wrapper method, the selection process is slow due to the involvement of the learning algorithm. The integration of multiple filter methods, IG, CS, and RF, with the OBWOA wrapper-based method, aims to leverage the strengths of both, resulting in a novel, optimised, and robust hybrid feature selection process that is unprecedented in the domain of SDP.

5.2.2 Defect Prediction based on Attention-based CNN

CNN is a deep neural network that efficiently handles image data. It expands the traditional artificial neural network by adding more layers, such as a convolutional layer, a pooling layer, a flattening layer, etc. A classic CNN applies to 2D data like images and videos; therefore, it is commonly referred to as 2D-CNN. Another recent development is 1D-CNN, a modified form of 2D-CNN [214, 218, 219]. 1D variant

implementation is feasible on a standard computer and comparatively faster due to fewer hidden layers and fewer parameters involved.

Fig. 5.2 illustrates the structure of the proposed 1D-CNN model with attention comprising the following important layers:

- **Input layer** - The 1D-input vector $K[n]$ where $n = 0, 1, 2, \ldots, N-1$ is given to the first layer of the CNN architecture. 'n' represents the software metrics present in the dataset. For example, the JM1 dataset contains 21 software metrics, so the input layer will accept 21 metrics.

- **Convolutional layer** - It is the foundational layer of 1D-CNN. A series of convolutions is performed using filters or kernels to acquire local features from the input data. The filters help detect deep semantic features from the low-level feature representation in the defect instances. We have used two convolutional layers with 64 and 32 filters and a kernel size of 1 for each filter.

- **ReLU Layer** - The output from the preceding layer (feature map) is subjected to a nonlinear activation function by the ReLU layer. It gives the model nonlinearity and enhances its capacity for expression.

- **Flatten Layer** - The feature maps produced as a result of the second convolutional layer are transformed into a single long continuous linear vector through the process of flattening.

- **Fully Connected (FC) Layer** - The output from the flattening layer is linearly transformed into a new representation through a set of weights and biases. To classify inputs accurately, we employed two FC layers. The first FC layer used ReLU activation, while the second FC layer used softmax activation, generating a probability ranging from 0 to 1.

- **Dropout Layer** - A regularisation approach randomly removes a particular proportion of the neurons in the FC layer during training. Hence, the model's generalization performance is enhanced and overfitting is prevented.
- **Attention Mechanism** - It is a technique commonly used in DL models to focus on the most important features or regions of an input, enabling the model to focus on relevant information and improve its performance. This mechanism has been effectively utilized in several computer vision tasks, like object recognition [220], segmentation [221], and detection [222].

In the context of SDP, attention mechanisms can be used in a CNN to learn discriminative features from software artefacts, such as source code or execution traces. This facilitates the identification of potential defects in a software system before they turn into failures or errors. We use the coordinate attention as described by [223]. The coordinate attention captures direction-aware and position-sensitive data that aids models in precisely locating and identifying the defects in software. As a computing unit, a coordinate attention block tries to increase the expressive power of the learned features. Any intermediate feature tensor, $X = [x_1, x_2, \ldots, x_C] \in R^{C \times H \times W}$, may be used as input; the transformed vector $Y = [y_1, y_2, \ldots, y_C]$, where X and Y are of the same size, is the output.

The channel attention algorithm modifies the global pooling operation by factorizing the pooling into two 1D feature encoding processes, one along the horizontal and one along the vertical coordinates. The input X undergoes encoding through two pooling kernels–$(H, 1)$ and $(1, W)$–capturing information along both horizontal and vertical coordinates for each channel. Therefore, Equation 5.2 can be used to express the output of the c-th channel at height h:

$$o_c^h(h) = \frac{1}{W} \sum_{i=0}^{W} x_c(h, i) \tag{5.2}$$

Likewise, the c-th channel's output at width w can be written as:

$$o_c^w(w) = \frac{1}{H} \sum_{j=0}^{H} x_c(j, w) \tag{3}$$

This allows the attention block to capture long-range spatial interactions while preserving precise positional information. This modification can be used in SDP to help 1D-CNN accurately locate defects of interest and improve the overall structure to capture deep semantic features.

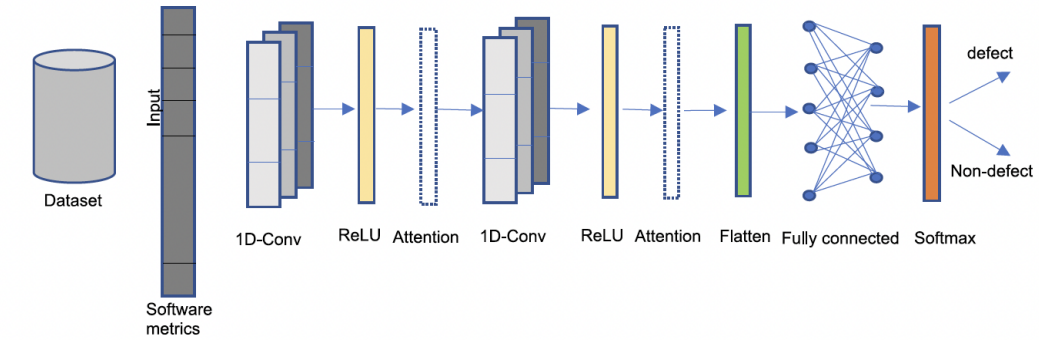

Figure 5.2: Architecture of attention-based 1D-CNN

5.3 Experimental Setup

This section discusses the datasets used, preprocessing steps, parameter settings, performance measures and the statistical tests used in this chapter.

5.3.1 Dataset collection and Preprocessing

We perform experiments on the 17 open-source datasets obtained from NASA, PROMISE, and the AEEEM repository, which provide publicly available benchmark datasets for the SDP problem to ensure that the research results can be verified,

replicated, and refuted. We have selected six datasets from NASA (JM1, KC1, MC1, PC3, PC4, and PC5), which contain traditional software metrics like McCabe, Halstead, Line Count, etc. Five datasets are selected from AEEEM (Equinox, Eclipse JDT Core, Eclipse PDE, Lucene, and Mylyn), which contain 61 measures and the PROMISE datasets (ant-1.7, camel-1.6, jEdit-4.3, poi-3.0, xalan-2.7, and xerces-1.4) contain 20 Object-oriented metrics, McCabe's metrics, and CK metrics. The details of the three datasets are given in Section ref subsec:dataset-details of Chapter 2.

After data collection, the data preprocessing steps were performed, which included three major steps: detecting missing values, handling outliers by median imputation, and tackling class imbalance by Random Oversampling. Finally, data standardisation was performed using min-max scaling.

5.3.2 Parameter Settings

For performing feature selection, only the training dataset is used in this paper. The model is evaluated using the same features that were selected from the training dataset. The filter methods – IG, CS, and RF are implemented using the Weka Tool (default settings), and top $\log_2 n$ features are selected. The wrapper-based, oppositional whale optimization algorithm and 1D-CNN model are implemented in Python 3.9 on Spyder 5.2.2. We have divided the data into 70% training and 30% testing before executing the CNN algorithm. Table 5.1 and Table 5.2 show the parameter settings for OBWOA and 1D-CNN.

5.3.3 Performance Metrics

To assess the effectiveness of predictive models for datasets with imbalanced class distributions, it is necessary to employ appropriate performance measures. In this chapter, we have considered AUC, F-measure, MCC, and G-mean, as these metrics

Table 5.1: Parameter settings for OBWOA

Parameters	**Values**
No. of iterations	100
No. of search agents (whales)	50
Spiral factor b	1
Convergence constant a	1
No. of runs	20

Table 5.2: Parameter settings for 1D-CNN

Parameters	**Values**
No. of convolutional layers	2
No. of dense layers	2
Activation function	ReLU / Softmax
Number of attention layers	2
Filter	64, 32
Kernel	1, 1
Optimizer	Adam
Learning rate	0.01
Loss function	Binary cross entropy
No. of epochs	250
Dropout	0.2

are widely utilized in the literature. The traditional measures, such as accuracy and precision, give biased results when handling imbalanced datasets [37]. Studies have advocated the use of AUC (Area under the Receiver's Operating Curve), a robust measure particularly effective for imbalanced and noisy datasets. These measures are explained in Section 2.7.7.

5.3.4 Statistical Analysis

The results of the chapter are statistically evaluated using the Wilcoxon Test to support and reinforce the findings and conclusions. Demšar [115] has suggested using non-parametric tests because assumptions regarding data distribution, such as normality and homogeneity, are relaxed. To evaluate the statistical significance of performance improvements, the Wilcoxon signed-rank test was applied to the CNN classifier across two categories: with and without feature selection (RQ1), and with and without the attention mechanism (RQ2). The full description of this test is mentioned in Section 2.7.8.

5.4 Result Analysis

The analysis of our results is structured around addressing the key research questions formulated at the outset of this study. Each research question was designed to evaluate specific aspects of the predictive modelling framework, feature selection techniques, and classifier performance in the context of software defect prediction. By aligning the results with these research questions, we have ensured a focused and systematic exploration of each objective.

5.4.1 Result Analysis based on RQ1

RQ1. How efficient is the SDP model developed using 1D-CNN with and without multi-filter wrapper feature selection?

The study utilized 17 software defect datasets to implement the 1D-CNN model with and without MFWFS. The model was designed with parameter settings specified in Table 5.2. The datasets are initially balanced using the random oversampling technique before training the model. The top $\log_2 n$ features are selected from IG, CS, and Relief-F methods applied to the training set. The OBWOA uses KNN ($K = 5$) as the internal classifier with parameter settings specified in Table 5.1. To handle the stochastic nature of OBWOA, we performed 20 runs of this technique to find the average results. Table 5.3 shows the original number of features in all datasets, the average number of selected features after performing OBWOA, the average selection ratio, the best fitness value, and the number of features in the MFWFS technique. For instance, the JM1 dataset has originally 21 features; we select the top $\log_2 n$ features, i.e., four features each from IG, CS, and RF methods, nine features are selected from OBWOA, and after their union, we get 12 features in MFWFS. We can observe from Table 5.3 that the attribute reduction rate varies from 20% to 77% across all seventeen datasets, resulting in a total average reduction of 50% of attributes using the above MFWFS technique. This implies that the new MFWFS technique generates feature subsets with substantially fewer features that produce higher-quality solutions.

The classification performance of the SDP model developed using the 1D-CNN defect predictor without and with MFWFS is evaluated across 17 software projects, measuring performance using F-measure, AUC, G-mean, and MCC. The results are displayed in Table 5.4, Table 5.5, Table 5.6, and Table 5.7. Results show that CNN with feature selection depicts the best F-measure, AUC, G-mean, and MCC values in nine, fifteen, ten, and ten out of seventeen datasets, respectively. There is an

Table 5.3: Average selected features obtained by MFWFS

Datasets	Original #features	#features (OBWOA)	Selection ratio	Best fitness value (OBWOA)	#features (IG + CS + RF + OBWOA)
JM1	21	9	0.4286	0.410967	12
MC1	38	5	0.1316	0.408940	13
KC1	21	8	0.3810	0.338598	15
PC3	37	7	0.1892	0.359309	16
PC4	37	6	0.1622	0.251322	16
PC5	38	12	0.3158	0.312920	21
EQ	61	4	0.0656	0.178979	14
JDT	61	15	0.2459	0.227630	24
PDE	61	9	0.1475	0.362150	22
Lucene	61	3	0.0492	0.321484	17
Mylyn	61	9	0.1475	0.342688	17
ant-1.7	20	6	0.3000	0.267295	12
camel-1.6	20	11	0.5500	0.379400	16
jedit-4.3	20	6	0.3000	0.410779	12
poi-3.0	20	9	0.4500	0.158550	14
Xalan-2.7	20	3	0.1500	0.158033	12
xerces-1.4	20	5	0.2500	0.086125	12

improvement in values of F-measure (1–8%), AUC (0.1–30%), G-mean (0.2–6%), and MCC (1–23.5%) for all the datasets except for the PDE and camel-1.6 datasets. The performance values of the PDE and camel-1.6 datasets after feature selection did not improve; hence, the feature selection was not helpful for those particular datasets.

Table 5.4: Comparative results of 1D-CNN with and without MFWFS based on F-measure

	JM1	**MC1**	**KC1**	**PC3**	**PC4**	**PC5**	**EQ**	**JDT**	**PDE**	**Lucene**	**Mylyn**	**Ant-1.7**	**Camel-1.6**	**Jedit-4.3**	**Poi-3.0**	**Xalan-2.7**	**Xerces-1.4**
Without FS	0.614	0.660	0.726	0.764	0.782	0.731	0.854	0.859	0.846	0.825	0.732	0.790	0.746	0.690	0.802	0.831	0.909
With FS	0.651	0.653	0.721	0.776	0.841	0.739	0.898	0.901	0.839	0.776	0.753	0.789	0.708	0.706	0.796	0.677	0.900

Table 5.5: Comparative results of 1D-CNN with and without MFWFS based on AUC

	JM1	**MC1**	**KC1**	**PC3**	**PC4**	**PC5**	**EQ**	**JDT**	**PDE**	**Lucene**	**Mylyn**	**Ant-1.7**	**Camel-1.6**	**Jedit-4.3**	**Poi-3.0**	**Xalan-2.7**	**Xerces-1.4**
Without FS	0.712	0.878	0.847	0.915	0.916	0.806	0.871	0.902	0.912	0.847	0.833	0.805	0.826	0.935	0.825	0.754	0.895
With FS	0.752	0.888	0.848	0.919	0.940	0.813	0.932	0.926	0.890	0.861	0.875	0.833	0.826	0.880	0.833	0.980	0.942

Table 5.6: Comparative results of 1D-CNN with and without MFWFS based on G-mean

	JM1	**MC1**	**KC1**	**PC3**	**PC4**	**PC5**	**EQ**	**JDT**	**PDE**	**Lucene**	**Mylyn**	**Ant-1.7**	**Camel-1.6**	**Jedit-4.3**	**Poi-3.0**	**Xalan-2.7**	**Xerces-1.4**
Without FS	0.662	0.865	0.821	0.910	0.890	0.771	0.864	0.891	0.904	0.848	0.827	0.813	0.802	0.888	0.828	0.393	0.899
With FS	0.701	0.858	0.804	0.906	0.914	0.775	0.914	0.914	0.883	0.863	0.853	0.815	0.789	0.855	0.824	0.969	0.917

Table 5.7: Comparative results of 1D-CNN with and without MFWFS based on MCC

	JM1	**MC1**	**KC1**	**PC3**	**PC4**	**PC5**	**EQ**	**JDT**	**PDE**	**Lucene**	**Mylyn**	**Ant-1.7**	**Camel-1.6**	**Jedit-4.3**	**Poi-3.0**	**Xalan-2.7**	**Xerces-1.4**
Without FS	0.281	0.406	0.511	0.610	0.623	0.496	0.717	0.732	0.713	0.668	0.515	0.590	0.530	0.466	0.637	0.227	0.822
With FS	0.347	0.475	0.491	0.624	0.709	0.506	0.815	0.806	0.671	0.586	0.560	0.588	0.479	0.471	0.627	0.467	0.807

To assess how the number of features affects classifier performance, the average F-measure, AUC, G-mean, and MCC values were compared using the Wilcoxon signed-rank test with a significance threshold of 0.05. The test results are presented in Table 5.8, showing the p-values for each comparison. A p-value below 0.05 suggests a statistically significant difference between the two techniques. The performance difference between the two techniques is denoted as either significantly different (S+) or not significantly different (NS), respectively. The analysis revealed a significant difference between the two methods in terms of AUC (p-value = 0.015) but not for F-measure, G-mean, or MCC.

This clearly implies that the impact of MFWFS on the 1D-CNN classifier is significant with respect to AUC values. Although 1D-CNN extracts high-level deep semantic features using various layers, feature selection using a hybrid filter-wrapper approach is applied prior to training the model, removing all the defective features that are irrelevant, redundant, and cause noise in the dataset and overfitting the model. As a result, this integration of static defect feature subsets with abstract deep semantic features in CNN improves the model's efficiency and builds a stronger ability to distinguish between defective/non-defective classes.

Table 5.8: Wilcoxon test result based on average F-measure, AUC, G-mean, and MCC

Comparison approach	**p-value (F-measure)**	**p-value (AUC)**	**p-value (G-mean)**	**p-value (MCC)**
CNN without feature selection / CNN with MFWFS	NS (0.463)	S (0.015)	NS (0.193)	NS (0.201)

5.4.2 Result Analysis based on RQ2

RQ2. How efficient is the SDP model developed using MFWFS-1D-CNN with and without attention?

The study evaluated the performance of SDP models created with MFWFS-1D-CNN, both with and without an attention layer, by measuring their F-measure, AUC, G-

mean, and MCC values. The results are presented in Table 5.9, Table 5.10, Table 5.11, and Table 5.12. Across all datasets, the F-measure, AUC, G-mean, and MCC values range from 0.613 to 0.901, 0.788 to 0.974, 0.761 to 0.958, and 0.381 to 0.897, respectively. Adding the attention layer improves the predictive model's performance in all the datasets except for one, i.e., *xalan*. We can clearly observe there is an improvement of about 0.1–11% in the values of F-measure, 0.5–11% in the values of AUC, 0.3–12% in the values of G-mean, and 0.12–35% in the values of MCC measures obtained after applying the attention layer in the 1D-CNN model. The improvement in the values is due to the application of the attention mechanism in the 1D-CNN model, which can detect the direction and position of the data, thus enabling the precise identification of the defects in the software.

Table 5.9: Comparative results of MFWFS-1D-CNN with and without attention based on F-measure

	JM1	**MC1**	**KC1**	**PC3**	**PC4**	**PC5**	**EQ**	**JDT**	**PDE**	**Lucene**	**Mylyn**	**Ant-1.7**	**Camel-1.6**	**Jedit-4.3**	**Poi-3.0**	**Xalan-2.7**	**Xerces-1.4**
Without attention	0.651	0.653	0.721	0.776	0.841	0.739	0.898	0.901	0.839	0.776	0.753	0.789	0.708	0.706	0.796	0.677	0.900
With attention	0.722	0.712	0.753	0.803	0.854	0.750	0.889	0.862	0.879	0.831	0.811	0.809	0.761	0.706	0.818	0.613	0.901

Table 5.10: Comparative results of MFWFS-1D-CNN with and without attention based on AUC

	JM1	**MC1**	**KC1**	**PC3**	**PC4**	**PC5**	**EQ**	**JDT**	**PDE**	**Lucene**	**Mylyn**	**Ant-1.7**	**Camel-1.6**	**Jedit-4.3**	**Poi-3.0**	**Xalan-2.7**	**Xerces-1.4**
Without attention	0.752	0.888	0.848	0.919	0.940	0.813	0.932	0.926	0.890	0.861	0.875	0.833	0.826	0.880	0.833	0.980	0.942
With attention	0.788	0.933	0.868	0.924	0.938	0.835	0.930	0.926	0.937	0.900	0.901	0.837	0.850	0.974	0.838	0.948	0.891

Table 5.11: Comparative results of MFWFS-1D-CNN with and without attention based on G-mean

	JM1	**MC1**	**KC1**	**PC3**	**PC4**	**PC5**	**EQ**	**JDT**	**PDE**	**Lucene**	**Mylyn**	**Ant-1.7**	**Camel-1.6**	**Jedit-4.3**	**Poi-3.0**	**Xalan-2.7**	**Xerces-1.4**
Without attention	0.701	0.858	0.804	0.906	0.914	0.775	0.914	0.914	0.883	0.863	0.853	0.815	0.789	0.855	0.824	0.969	0.917
With attention	0.761	0.921	0.834	0.911	0.917	0.790	0.897	0.896	0.927	0.881	0.876	0.822	0.816	0.958	0.835	0.935	0.890

Table 5.12: Comparative results of MFWFS-1D-CNN with and without attention based on MCC

	JM1	**MC1**	**KC1**	**PC3**	**PC4**	**PC5**	**EQ**	**JDT**	**PDE**	**Lucene**	**Mylyn**	**Ant-1.7**	**Camel-1.6**	**Jedit-4.3**	**Poi-3.0**	**Xalan-2.7**	**Xerces-1.4**
Without attention	0.347	0.475	0.491	0.624	0.709	0.506	0.815	0.806	0.671	0.586	0.560	0.588	0.479	0.471	0.627	0.467	0.807
With attention	0.469	0.509	0.551	0.667	0.730	0.534	0.897	0.735	0.773	0.678	0.646	0.623	0.555	0.517	0.649	0.381	0.808

To statistically validate our results, we performed the Wilcoxon test on all four performance measures. Table 5.13 represents the p-values obtained when the MFWFS-1D-CNN-attention model is compared with MFWFS-1D-CNN. The p-values obtained with respect to F-measure = 0.03, AUC = 0.041, and MCC = 0.012 show a significant difference in the two approaches, but not significantly with respect to the G-mean measure. These results suggest that the MFWFS-1D-CNN attention model outperforms the MFWFS-1D-CNN model without attention, signifying the importance of the attention layer in the neural network.

Table 5.13: Wilcoxon test result based on F-measure, AUC, G-mean, and MCC

Comparison approach	**p-value (F-measure)**	**p-value (AUC)**	**p-value (G-mean)**	**p-value (MCC)**
MFWFS-1D-CNN without attention / MFWFS-1D-CNN with attention	S+ (0.030)	S+ (0.041)	NS (0.061)	S+ (0.012)

5.4.3 Result Analysis based on RQ3

RQ3. How effective are the SDP models developed using the proposed hybrid model MFWFS-1D-CNN-Attention in comparison to the models created with individual filters and a wrapper-based CNN model?

To assess whether the proposed MFWFS-1D-CNN-Attention model is able to effectively classify the software defects, we made a comparison of this model with the sole wrapper method (OBWOA) and combinations of filter (IG, CS, RF) and wrapper methods. Table 5.14 reports the results using the four performance measures - F-measure, AUC, G-mean, and MCC.

Table 5.14: Performance of different feature selection methods on 1D-CNN

Datasets	**Methodology**	**F-measure**	**AUC**	**G-mean**	**MCC**
JM1	OBWOA	0.632	0.734	0.682	0.315

Datasets	Methodology	F-measure	AUC	G-mean	MCC
	IG + OBWOA	0.632	0.730	0.680	0.314
	CS + OBWOA	0.616	0.712	0.659	0.279
	RF + OBWOA	0.638	0.730	0.688	0.325
	IG + CS + RF + OBWOA	**0.722**	**0.788**	**0.761**	**0.469**
MC1	OBWOA	0.581	0.836	0.815	0.289
	IG + OBWOA	0.585	0.872	0.860	0.313
	CS + OBWOA	0.594	0.852	0.842	0.315
	RF + OBWOA	0.592	0.887	0.885	0.329
	IG + CS + RF + OBWOA	**0.712**	**0.933**	**0.921**	**0.509**
KC1	OBWOA	0.709	0.834	0.780	0.463
	IG + OBWOA	0.701	0.832	0.767	0.444
	CS + OBWOA	0.660	0.845	0.757	0.420
	RF + OBWOA	0.672	0.821	0.760	0.412
	IG + CS + RF + OBWOA	**0.753**	**0.868**	**0.834**	**0.551**
PC3	OBWOA	0.691	0.827	0.824	0.469
	IG + OBWOA	0.697	0.856	0.849	0.512
	CS + OBWOA	0.721	0.850	0.854	0.523
	RF + OBWOA	0.732	0.890	0.868	0.547
	IG + CS + RF + OBWOA	**0.803**	**0.924**	**0.911**	**0.667**
PC4	OBWOA	0.745	0.891	0.861	0.556
	IG + OBWOA	0.794	0.923	0.901	0.641
	CS + OBWOA	0.788	0.864	0.849	0.598
	RF + OBWOA	0.811	0.925	0.906	0.664
	IG + CS + RF + OBWOA	**0.854**	**0.938**	**0.917**	**0.730**
PC5	OBWOA	0.692	0.796	0.730	0.422
	IG + OBWOA	0.680	0.778	0.719	0.402
	CS + OBWOA	0.696	0.789	0.735	0.431
	RF + OBWOA	0.702	0.798	0.727	0.427

Datasets	Methodology	F-measure	AUC	G-mean	MCC
	IG + CS + RF + OBWOA	**0.750**	**0.835**	**0.790**	**0.534**
EQ	OBWOA	0.780	0.816	0.796	0.583
	IG + OBWOA	0.806	0.862	0.833	0.658
	CS + OBWOA	0.797	0.845	0.815	0.624
	RF + OBWOA	0.840	0.879	0.856	0.701
	IG + CS + RF + OBWOA	**0.889**	**0.930**	**0.897**	**0.897**
JDT	OBWOA	0.824	0.881	0.866	0.672
	IG + OBWOA	0.861	0.897	0.892	0.733
	CS + OBWOA	0.841	0.882	0.874	0.697
	RF + OBWOA	0.855	0.897	0.887	0.724
	IG + CS + RF + OBWOA	**0.862**	**0.926**	**0.896**	**0.735**
PDE	OBWOA	0.734	0.827	0.805	0.505
	IG + OBWOA	0.805	0.846	0.848	0.624
	CS + OBWOA	0.799	0.861	0.853	0.619
	RF + OBWOA	0.778	0.841	0.843	0.583
	IG + CS + RF + OBWOA	**0.879**	**0.937**	**0.927**	**0.773**
Lucene	OBWOA	0.613	0.732	0.720	0.305
	IG + OBWOA	0.704	0.864	0.828	0.475
	CS + OBWOA	0.673	0.771	0.747	0.385
	RF + OBWOA	0.708	0.835	0.828	0.480
	IG + CS + RF + OBWOA	**0.831**	**0.900**	**0.881**	**0.678**
Mylyn	OBWOA	0.773	0.864	0.849	0.578
	IG + OBWOA	0.757	0.863	0.829	0.546
	CS + OBWOA	0.751	0.851	0.835	0.540
	RF + OBWOA	0.738	0.837	0.821	0.517
	IG + CS + RF + OBWOA	**0.811**	**0.901**	**0.876**	**0.646**
ant-1.7	OBWOA	0.771	0.837	0.798	0.555
	IG + OBWOA	0.757	0.831	0.784	0.527

Datasets	Methodology	F-measure	AUC	G-mean	MCC
	CS + OBWOA	0.786	0.839	0.813	0.585
	RF + OBWOA	0.794	0.821	0.811	0.601
	IG + CS + RF + OBWOA	**0.809**	**0.837**	**0.822**	**0.623**
camel-1.6	OBWOA	0.578	0.739	0.668	0.271
	IG + OBWOA	0.698	0.810	0.773	0.460
	CS + OBWOA	0.681	0.809	0.754	0.424
	RF + OBWOA	0.689	0.790	0.750	0.424
	IG + CS + RF + OBWOA	**0.761**	**0.850**	**0.816**	**0.555**
jedit-4.3	OBWOA	**0.761**	0.970	0.869	**0.558**
	IG + OBWOA	0.669	0.958	0.942	0.460
	CS + OBWOA	0.741	0.942	0.871	0.525
	RF + OBWOA	0.652	0.960	0.945	0.446
	IG + CS + RF + OBWOA	0.722	**0.982**	**0.964**	0.544
poi-3.0	OBWOA	0.774	0.797	0.794	0.574
	IG + OBWOA	0.795	0.822	0.820	0.619
	CS + OBWOA	0.800	0.821	0.818	0.641
	RF + OBWOA	0.807	0.833	0.832	0.640
	IG + CS + RF + OBWOA	**0.818**	**0.838**	**0.835**	**0.649**
Xalan-2.7	OBWOA	0.687	**0.989**	**0.970**	0.482
	IG + OBWOA	**0.755**	0.879	0.873	**0.554**
	CS + OBWOA	0.687	0.974	0.800	0.403
	RF + OBWOA	0.612	0.970	0.815	0.323
	IG + CS + RF + OBWOA	0.613	0.948	0.935	0.381
xerces-1.4	OBWOA	0.873	0.873	0.863	0.750
	IG + OBWOA	**0.912**	**0.917**	**0.907**	**0.827**
	CS + OBWOA	0.873	0.874	0.872	0.748
	RF + OBWOA	0.873	0.872	0.870	0.749
	IG + CS + RF + OBWOA	0.901	0.891	0.890	0.808

In terms of all the evaluation measures, MFWFS (IG+CS+RF+OBWOA) on the CNN attention model produced the highest values in JM1, MC1, KC1, PC1, PC3, PC4, PC5, EQ, JDT, Lucene, Mylyn, ant-1.7, camel-1.6, and poi-3.0 datasets. However, for the jedit-4.3 dataset, the performance of the IG+CS+RF+OBWOA method was better in terms of AUC and G-mean, while the OBWOA wrapper method alone was better in terms of F-measure and MCC scores. For Xalan-2.7, IG+OBWOA provided the best F-measure and MCC value, while OBWOA gave the best performance in terms of AUC and G-mean. For the Xerces-1.4 dataset, IG+OBWOA performed the best, with the highest values for all evaluation measures.

We also present the boxplots to visually compare the difference in the prediction performances between OBWOA, IG+OBWOA, CS+OBWOA, RF+OBWOA, and IG+CS+RF+OBWOA over seventeen datasets, with respect to four performance measures in Fig. 5.3. The x-axis denotes the different methodologies being compared, and the y-axis denotes the four evaluation measures. We observe that for each of the four evaluation metrics, the median value achieved by IG+CS+RF+OBWOA is higher than that obtained by other approaches. The average predicted F-measure value by IG+CS+RF+OBWOA is 0.794 (for all the datasets), which is an improvement between 7% (IG+OBWOA) and 10.41% (OBWOA). The average AUC (0.896) by IG+CS+RF+OBWOA obtains improvement between 4.4% (RF+OBWOA) and 7% (OBWOA). The average G-mean (0.875) by IG+CS+RF+OBWOA obtains improvement between 5.4% (IG+OBWOA) and 8.65% (OBWOA). The average MCC (0.632) by IG+CS+RF+OBWOA obtains improvement between 18% (IG+OBWOA) and 28.78% (OBWOA).

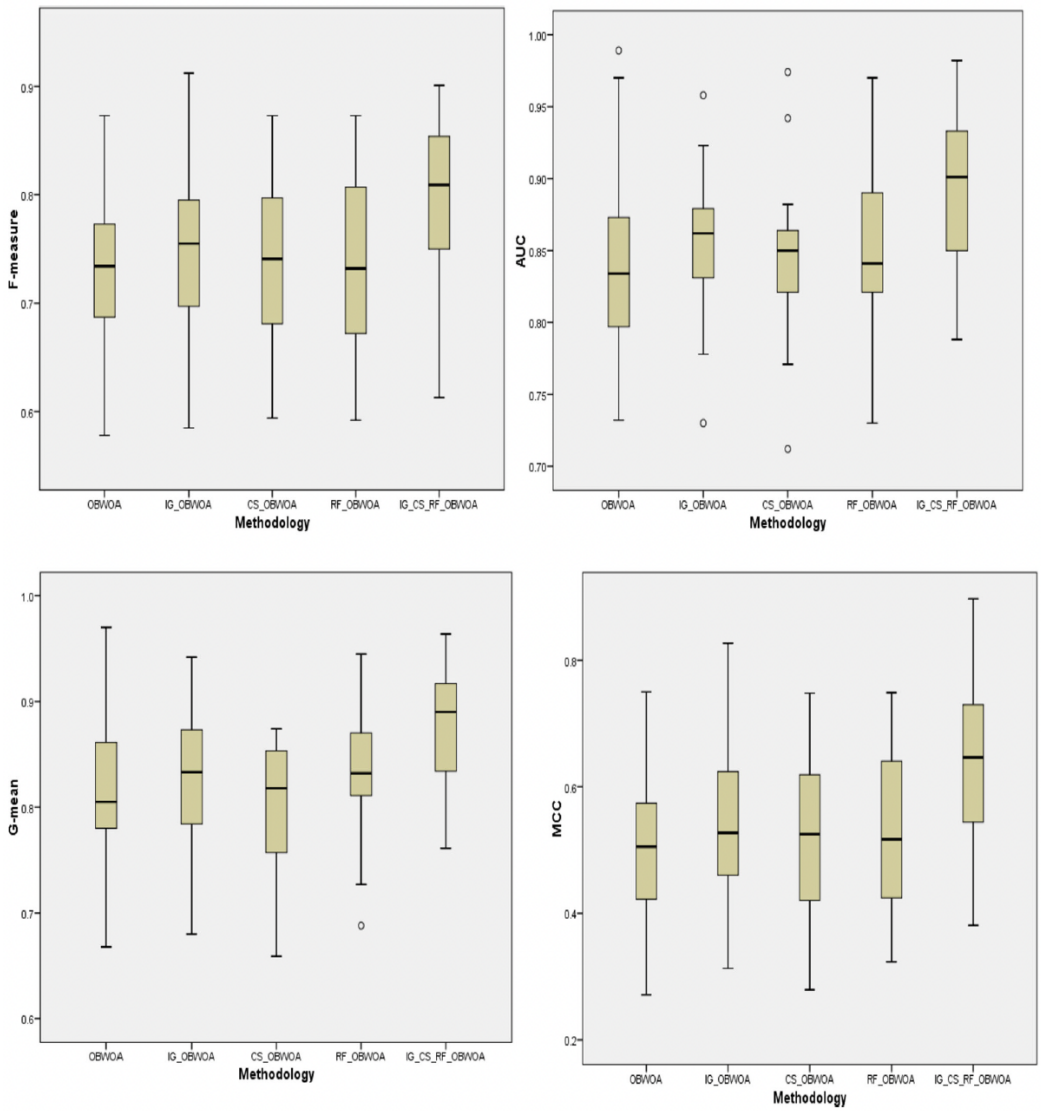

Figure 5.3: Box plot for F-measure, AUC, G-mean, and MCC values for different feature selection methodologies

We also compared the performance of the proposed feature selection method IG+CS+RF+OBWOA with the transformer-based model BERT [224] used in a previous literature study to validate the performance and robustness of the proposed method against the original 1D-CNN-attention based model. The F-measure, AUC, G-mean, and MCC values are reported in Table 5.15. The results indicate that

the IG+CS+RF+OBWOA with the 1D-CNN model consistently outperforms the IG+CS+RF+OBWOA with the BERT approach across all datasets in terms of F-measure, AUC, G-mean, and MCC values.

Higher F-measure values for 1D-CNN indicate better balance between precision and recall, which is essential for accurate defect prediction. AUC values are also consistently higher for 1D-CNN, indicating stronger discriminatory power and better classification ability. Similarly, G-mean values show that 1D-CNN is better at controlling the trade-off between sensitivity and specificity, resulting in more balanced classification results. Additionally, MCC values are notably higher for CNN, which provides insight into the overall quality of predictions, taking into account both positive and negative classes. This indicates a stronger correlation between predicted and actual classes compared to BERT. Although BERT-like transformers generally perform well in handling complex data patterns and have successful results in existing literature studies, in this specific case, IG+CS+RF+OBWOA with 1D-CNN attention combination leads to enhanced performance across these critical metrics, making it a more accurate model for predicting software defects.

In conclusion, we can say the IG+CS+RF+OBWOA-based CNN attention model behaves consistently over all the considered datasets, mainly performing better than other combinations of the filter and wrapper methods. Since all of the measures - F-measure, AUC, G-mean, and MCC-show significant improvement in the results, we can justify the use of this combination of feature selection techniques for SDP.

5.4.4 Result Analysis based on RQ4

RQ4. How effective is the SDP model developed using the proposed hybrid classifier (MFWFS-1D-CNN-Attention) in comparison with other state-of-the-art ML and hybrid techniques?

Table 5.15: Performance comparison of IG + CS + RF + OBWOA with CNN and BERT

Datasets	Methodology	F-measure	AUC	G-mean	MCC
JM1	IG + CS + RF + OBWOA with CNN	**0.722**	**0.788**	**0.761**	**0.469**
	IG + CS + RF + OBWOA with BERT	0.516	0.571	0.503	0.287
MC1	IG + CS + RF + OBWOA with CNN	**0.712**	**0.933**	**0.921**	**0.509**
	IG + CS + RF + OBWOA with BERT	0.579	0.500	0.235	0.117
KC1	IG + CS + RF + OBWOA with CNN	**0.753**	**0.868**	**0.834**	**0.551**
	IG + CS + RF + OBWOA with BERT	0.611	0.818	0.521	0.475
PC3	IG + CS + RF + OBWOA with CNN	**0.803**	**0.924**	**0.911**	**0.667**
	IG + CS + RF + OBWOA with BERT	0.470	0.580	0.569	0.180
PC4	IG + CS + RF + OBWOA with CNN	**0.854**	**0.938**	**0.917**	**0.730**
	IG + CS + RF + OBWOA with BERT	0.381	0.589	0.530	0.153
PC5	IG + CS + RF + OBWOA with CNN	**0.750**	**0.835**	**0.790**	**0.534**
	IG + CS + RF + OBWOA with BERT	0.463	0.562	0.534	0.173
EQ	IG + CS + RF + OBWOA with CNN	**0.889**	**0.930**	**0.897**	**0.897**
	IG + CS + RF + OBWOA with BERT	0.677	0.792	0.673	0.509
JDT	IG + CS + RF + OBWOA with CNN	**0.862**	**0.926**	**0.896**	**0.735**
	IG + CS + RF + OBWOA with BERT	0.861	0.914	0.805	0.729
PDE	IG + CS + RF + OBWOA with CNN	**0.879**	**0.937**	**0.927**	**0.773**
	IG + CS + RF + OBWOA with BERT	0.388	0.557	0.463	0.119
Lucene	IG + CS + RF + OBWOA with CNN	**0.831**	**0.900**	**0.881**	**0.678**
	IG + CS + RF + OBWOA with BERT	0.456	0.763	0.617	0.272
Mylyn	IG + CS + RF + OBWOA with CNN	**0.811**	**0.901**	**0.876**	**0.646**
	IG + CS + RF + OBWOA with BERT	0.393	0.615	0.554	0.155
Ant-1.7	IG + CS + RF + OBWOA with CNN	**0.809**	**0.837**	**0.822**	**0.623**
	IG + CS + RF + OBWOA with BERT	0.452	0.752	0.621	0.304
Camel-1.6	IG + CS + RF + OBWOA with CNN	**0.761**	**0.850**	**0.816**	**0.555**
	IG + CS + RF + OBWOA with BERT	0.314	0.684	0.563	0.218
Jedit-4.3	IG + CS + RF + OBWOA with CNN	**0.722**	**0.982**	**0.964**	**0.544**
	IG + CS + RF + OBWOA with BERT	0.650	0.980	0.703	0.393
Poi-3.0	IG + CS + RF + OBWOA with CNN	0.818	0.838	0.835	0.649
	IG + CS + RF + OBWOA with BERT	**0.851**	**0.912**	**0.874**	**0.732**
Xalan-2.7	IG + CS + RF + OBWOA with CNN	0.613	**0.948**	**0.935**	0.381
	IG + CS + RF + OBWOA with BERT	**0.643**	0.874	0.723	**0.548**
Xerces-1.4	IG + CS + RF + OBWOA with CNN	**0.901**	**0.891**	**0.890**	**0.808**
	IG + CS + RF + OBWOA with BERT	0.854	0.832	0.819	0.747

In order to assess the effectiveness of the proposed approach (MFWFS-1D-CNN-Attention), we conducted a comparison of its results with those of other state-of-the-art techniques that are currently available in the literature. We report the results of only those techniques evaluated on the same datasets used in our study and a common performance measure (AUC), since it is commonly available in the literature studies. Table 5.16 provides a list of the abbreviated comparison algorithms along with their particular implementation/parameter values utilized in evaluating the study.

Table 5.17 compares the proposed algorithm MFWFS-1D-CNN-Attention and other algorithms listed in Table 5.16, with respect to the AUC performance measure. The best values are highlighted in bold. We observe that the proposed approach achieves the highest values of the AUC measure in seven datasets (KC1, PC3, camel-1.6, jedit-4.3, poi-3.0, xalan-2.7, xerces-1.4) out of the ten datasets used for comparison with eighteen algorithms, and second-best in the other three datasets (JM1, PC4, and ant-1.7). For the JM1 dataset, PCA-ANN obtained the highest AUC value of 0.81. For the PC4 dataset, RF and PCA-ANN obtained the best AUC value of 0.97. For the ant-1.7 dataset, ANN obtained the best AUC value of 0.847.

Table 5.16: Abbreviations of the algorithms used for comparison with the proposed model

Abbr.	**Algorithm**	**References**	**Implementation/parameter values**
k-NN	k-nearest neighbour	[203, 207]	The number of neighbours was varied between 1, 3, 5, up to 15
SVM	Support vector machine	[203, 207]	Kernel used-RBF, A multilevel grid search was employed to tune kernel width and regularization parameter C, ranging from $\log(C) = [-6, -5, \ldots, 20]$

Abbr.	Algorithm	References	Implementation/parameter values
RF	Random forest	[203]	Number of trees tuned between 10, 50, 100, 250, 500, and 1000, and the number of attributes selected per tree as $\{0.5, 1, 2\} \times \sqrt{M}$, where M is the number of attributes in the dataset
L-SVM	Lagrangian support vector machine	[203, 207]	No kernel function used, A range from $\log(C) = [6, 5, \ldots, 20]$ has been evaluated
LS-SVM	Least squares support vector machine	[203, 207]	Kernel - RBF, A multilevel grid search was employed to tune kernel width and regularization parameter C, ranging from $\log(C) = [-6, -5, \ldots, 20]$
NB	Naïve Bayes	[207]	Default (Gaussian NB)
LDA	Linear discriminant analysis	[203, 207]	Solver: SVD, shrinkage: None
C4.5	Decision tree	[203]	Pruning strategies varied confidence levels from 0.05 to 0.7, with and without Laplacian smoothing and subtree raising
ANN	Artificial neural network	[18]	Three-layer network, hidden neurons (10), no. of epochs (300), activation function (sigmoid), learning rate (0.1)
CNN-WSHCKE	CNN-whale optimization-simulated annealing-based kernel extreme machine learning	[62]	Number of neurons in convolutional layers: 32, 64, 16, kernel sizes: 3×3, 3×3, and 4×4, activation function: ReLU, KELM hyperparameters: Gaussian kernel's bandwidth σ
SSA-BPNN	Salp swarm algorithm-backpropagation neural network	[65]	SSA: Population size-30, max iterations-300. BPNN: three-layer network, hidden layer: 1, hidden neurons: $2n + 1$

Abbr.	Algorithm	References	Implementation/parameter values
PCA-ANN	Principal component analysis-ANN	[207]	PCA uses maximum likelihood estimation, ANN: hidden neurons $2n+1$, sigmoid activation function, minimizes MSE during training
LRNN	Layered recurrent neural network	[198]	No. of iterations: 1000, input layer neurons: no. of features, hidden layer neurons: number of features/2, output neurons: 1, threshold: 0.5
BGA-BPSO-BACO-LRNN	Binary genetic algorithm-binary particle swarm optimization-binary ant colony optimization-layered recurrent neural network	[198]	BGA: Iterations: 300, Population: 40, Crossover rate: 0.7, Mutation rate: 0.1, Selection: Roulette wheel, Crossover: Single, double, or uniform (random). BPSO: Iterations: 300, Swarm: 40, $c1 = c2 = 1.5$, Inertia weight: 0.8, Velocity range: [0,1]. BACO: Iterations: 300, Ants: 20, Initial pheromone: 1, Pheromone weight (α): 0.8, Heuristic weight (β): 0.8, Evaporation rate: 0.6
TBWOA-DT	Tournament selection method with binary Whale optimization algorithm-decision tree	[55]	Population size: 10, Iterations: 100, Tournament size (T): 3
EBMFOV3	Enhanced binary moth flame optimization	[145]	Population size: 20-50, Iterations: 100, Transfer functions used: S-shaped and V-shaped
DP-ARNN	Defect prediction via attention-based recurrent neural network	[225]	Embedding dimension: 30, AST vector length: 2000 Bi-LSTM units: 40 per layer, First hidden layer nodes: 16, Second hidden layer nodes: 24, Batch size: 32, Epochs: 20, Activation functions: tanh in the first layer, linear in the second layer, sigmoid in the output layer, Loss function: Binary Cross-Entropy Optimizer: RMSprop

Abbr.	Algorithm	References	Implementation/parameter values
CNN	CNN	[225]	Number of filters: 10, Filter length: 5, Fully connected layer nodes: 100
RNN	Recurrent neural network	[225]	Same as for DP-ARNN [225]

Table 5.17: Results of the proposed approach against eighteen other algorithms in respect of AUC

Techniques	JM1	KC1	PC3	PC4	ant-1.7	camel-1.6	jedit-4.3	poi-3.0	xalan-2.7	xerces-1.4
KNN	0.71	0.70	0.77	0.87	–	–	–	–	–	–
SVM	0.72	0.76	0.77	0.92	–	–	–	–	–	–
RF	0.76	0.78	0.82	**0.97**	–	0.677	0.797	0.636	0.674	0.576
L-SVM	0.73	0.76	0.84	0.92	–	–	–	–	–	–
LS-SVM	0.74	0.77	0.83	0.94	–	–	–	–	–	–
NB	0.69	0.76	0.81	0.84	–	–	–	–	–	–
LDA	0.73	0.78	0.82	0.88	–	–	–	–	–	–
C4.5	0.72	0.71	0.78	0.93	–	–	–	–	–	–
ANN	0.713	0.735	–	–	**0.847**	0.681	0.461	–	0.817	–
SSA-BPNN	0.70	0.79	0.87	0.90	0.79	–	0.97	–	–	–
PCA-ANN	**0.81**	0.79	0.89	**0.97**	–	–	–	–	–	–
LRNN	–	–	–	–	0.605	0.542	0.696	–	0.443	–
BGA-BPSO-BACO-LRNN	–	–	–	–	0.523	0.505	0.684	–	0.42	–
TBWOA-DT	–	–	–	–	0.737	0.677	0.847	–	0.796	–
EBMFOV3	–	–	–	–	0.762	0.658	0.750	–	0.846	–
DP-ARNN	–	–	–	–	–	0.790	0.820	0.796	0.674	0.761
CNN	–	–	–	–	–	0.732	0.841	0.745	0.674	0.671
RNN	–	–	–	–	–	0.766	0.842	0.764	0.654	0.730
Proposed approach	0.788	**0.868**	**0.924**	**0.938**	**0.837**	**0.850**	**0.974**	**0.838**	**0.948**	**0.891**

Bold values represent the highest values among all the techniques for each dataset

We have also compared MFWFS-1D-CNN-Attention with state-of-the-art models including BERT [224], CodeBERT [226] transformer models used in previous studies for software defect prediction in Table 7.2. The average performance values are recorded across all three datasets - NASA, AEEEM, and PROMISE. Our proposed approach consistently outperformed all other models in terms of F-measure, AUC, G-mean, and MCC, demonstrating its superiority. In the case of AEEEM datasets, an impressive F-measure of 0.855 is achieved, which is substantially higher than the values obtained by CNN-WSHCKE (0.691), BERT (0.540), and CodeBERT (0.32). Similarly, in the case of PROMISE datasets, a remarkable F-measure of 0.768 is obtained, outperforming the F-measure values of CNN-WSHCKE (0.485), BERT (0.652), and CodeBERT (0.306). Furthermore, our approach exhibits notable improvements in AUC, G-mean, and MCC metrics across all datasets compared to the

other models.

In a nutshell, we can say that the MFWFS-1D-CNN-based attention model has a powerful ability to address the SDP issue. In order to obtain the best feature subset for each software project, the unique properties of the wrapper method OBWOA are integrated with IG+CS+RF filter methods, significantly outperforming the use of these techniques in isolation. Further, applying a deep learning-based CNN model with an attention mechanism results in the identification of crucial features that improve the performance of SDP.

Table 5.18: Average performance values of the proposed approach against other models

Dataset	**Proposed Approach**				**CNN-WSHCKE**				**BERT**				**CodeBERT**			
	F-measure	AUC	G-mean	MCC	F-measure	AUC	G-mean	MCC	F-measure	AUC	G-mean	MCC	F-measure	AUC	G-mean	MCC
AEEEM	0.855	0.919	0.895	0.746	0.691	0.792	0.799	0.484	0.540	0.730	0.652	0.300	0.320	0.497	0.425	0.000
PROMISE	0.768	0.890	0.876	0.589	0.485	0.744	0.646	0.353	0.652	0.883	0.685	0.407	0.306	0.500	0.500	0.000
NASA	0.766	0.881	0.856	0.577	–	–	–	–	0.349	0.548	0.448	0.002	0.355	0.503	0.396	0.000

5.5 Discussion

This study proposed a novel hybrid approach for SDP that combines multi-filter wrapper feature selection (MFWFS) with a one-dimensional Convolutional Neural Network (1D-CNN) enhanced by an attention mechanism. The experiments were systematically designed and evaluated across 17 open-source software defect datasets from NASA, AEEEM, and PROMISE to validate the effectiveness of this approach and offer several important insights.

1. The multi-filter wrapper feature selection technique successfully addressed the issue of high-dimensional, noisy, and redundant feature sets. By integrating Information Gain (IG), Chi-square (CS), and Relief-F filter methods with the Opposition-Based Whale Optimization Algorithm (OBWOA), the approach achieved a significant average feature reduction of nearly 50%. The synergy between filter and wrapper methods ensured computational efficiency without compromising classification performance.

2. The MFWFS applied to the CNN model showed superior performance compared to no feature selection, particularly in terms of the ROC-AUC metric. The Wilcoxon Signed Rank test confirmed that applying MFWFS to CNN models led to statistically significant improvements in AUC ($p = 0.015$). This suggests that the hybrid feature selection primarily strengthens the classifier's discrimination capability, particularly in imbalanced settings.

3. The 1D-CNN architecture outperformed traditional 2D-CNNs due to its lower computational complexity. Integrating the coordinate attention mechanism into the 1D-CNN architecture further improved classification performance. This enhancement was consistent across nearly all datasets, with improvements ranging from 0.1% to 11% in F-measure and up to 35% in MCC.

4. The proposed model outperformed 18 existing ML and hybrid models- including SVM, Random Forest, PCA-ANN, and transformer-based models like BERT and CodeBERT- on key performance metrics across multiple datasets. Notably, the proposed approach achieved the highest AUC in 7 out of 10 datasets compared and was the second-best in the remaining ones.

In conclusion, the integration of a robust feature selection strategy with a deep learning architecture shows consistently high predictive performance across a range of benchmark datasets. These findings underscore the model's capability not only to enhance classification performance but also to contribute meaningfully to ongoing research in feature selection and optimization methodologies within the domain of software engineering.

Chapter 6

Hybrid Model for SDP using parameter tuning of ML classifier

6.1 Introduction

Machine learning algorithms like decision trees, random forests, logistic regression, and neural networks have been employed over the years for SDP. Some studies have shown that the SVM classifier in SDP works very well, as it is suitable for binary classification tasks predicting software modules as defective/non-defective [134]. Due to its high predictive capability and non-linear computing power, SVM can successfully capture complex relationships between software attributes and defect-proneness with the right choice of relevant software metrics as input features. However, earlier research shows that the performance of SVM is greatly influenced by its parameters, which were traditionally tuned using trial-and-error methods. This approach is time-consuming, labor-intensive, and prone to inadequate parameter settings due to complex software systems and the huge range of possible parameter combinations.

Hyperparameter tuning plays a vital role in improving the effectiveness of SDP

models. Models trained with default hyperparameters often fail to capture the complexity of real-world software data, leading to suboptimal performance. This challenge becomes even more pronounced when dealing with imbalanced datasets, which are common in defect prediction. Properly tuned hyperparameters enable the model to learn more effectively, balance predictive power across classes, and ultimately deliver more reliable results.

In this chapter, we investigate two metaheuristic algorithms to automatically tune the parameters of the SVM classifier for SDP. Metaheuristic algorithms are optimization techniques that draw their inspiration from natural processes. These algorithms are suitable for solving complex optimization problems by finding near-optimal solutions through efficiently exploring large input search spaces. We aim to enhance the SVM classifier's performance by fine-tuning its parameters using two algorithms: GWO [90] and SSO [99]. This chapter determines an optimal combination of C and γ parameters using GWO and SSO metaheuristic optimization techniques to obtain strong classification performance of the SVM model. The prediction capability of the hybrid models GWO-SVM and SSO-SVM is evaluated and compared, and the results reveal improved performance over the classic SVM model. The models are validated on five open-source software projects from the AEEEM repository. As the datasets are imbalanced in nature, we apply the oversampling technique to balance the training data before developing these models. Additionally, we employ the 10-fold cross-validation technique while training the model to avoid overfitting.

We have designed the following research questions to achieve the above objectives:

- *RQ1: How does parameter optimization using GWO and SSO impact the performance of the SVM classifier in SDP using the AEEEM datasets?*

  The primary objective of RQ1 is to evaluate whether parameter optimization using GWO and SSO can significantly improve the performance of SVM classi-

fiers in SDP. We have compared the performance measure scores of F-measure and AUC across AEEEM datasets obtained before and after optimization. This demonstrates the impact of metaheuristic tuning on enhancing prediction accuracy and robustness of SVM models.

- *RQ2: What are the optimal parameter settings obtained through GWO and SSO for SVM classifiers in SDP?*

  The primary objective of RQ2 is to identify the optimal hyperparameter settings (C and γ) for SVM classifiers by using GWO and SSO. By leveraging these metaheuristic techniques, the study aims to maximize classification performance and determine the most effective parameter values across different datasets, ensuring robust and well-tuned SVM models.

- *RQ3: How do the optimized SVM models using GWO and SSO compare with Random Search and Grid Search techniques in SDP?*

  The primary objective of RQ3 is to evaluate whether metaheuristic-based SVM models (GWO-SVM and SSO-SVM) outperform traditional tuning methods like Random Search and Grid Search in SDP. By comparing their F-measure and AUC scores and validating results statistically with the Friedman test, the study aims to establish the superiority and significance of GWO and SSO in enhancing SVM performance across diverse datasets.

The paper is arranged as follows: Section 2 briefly describes the methodology, including dataset collection, preprocessing steps, model development and validation steps, performance measures, and statistical tests used. Section 3 evaluates the performance of the hybrid algorithms and discusses the results. Section 4 provides the discussion of the chapter. The results of this chapter are presented at a conference in [227].

6.2 Research Methodology

6.2.1 Dataset Collection and Variables

We used five openly accessible and frequently used software projects from AEEEM repository- EQ, JDT, PDE, Lucene, and Mylyn, to be utilized as the benchmark dataset for SDP. These datasets have been widely employed in empirical software engineering research in order to analyse and compare different predictive models and fault prediction methodologies. The datasets comprise 61 software metrics derived from source code measurements, including change metrics, entropy metrics and previous defects. Section 2.7.2 contains the detailed description of the datasets.

6.2.2 Data Pre-processing

It is imperative to first normalise the input dataset before beginning the training process to produce reliable results, as each dimension may have a different range of values. We use the min-max normalization method to make all input values lie in the range [0,1].

To avoid poor model training due to imbalanced datasets, we use a random oversampling method that produces a training set that is balanced by replicating the instances from the minority class randomly [228].

6.2.3 Model Development and validation

We devise an SDP model that can help efficiently identify modules into defective and non-defective classes using two bio-inspired metaheuristic algorithms - GWO and SSO- to fine-tune the C and gamma parameters of the SVM classifier (RBF Kernel). Fig. 6.1 explains a step-by-step procedure followed in our research for model

prediction. Each optimization algorithm was run for 100 iterations with a population size of 30, and the objective was to maximize the AUC (Area Under the Receiver Operating Characteristic Curve) of the SVM classifier. After the predictive models are developed, we analyze their performance with metrics like AUC and F-measure. The predictive models are compared to other traditional parameter tuning techniques using statistical tests.

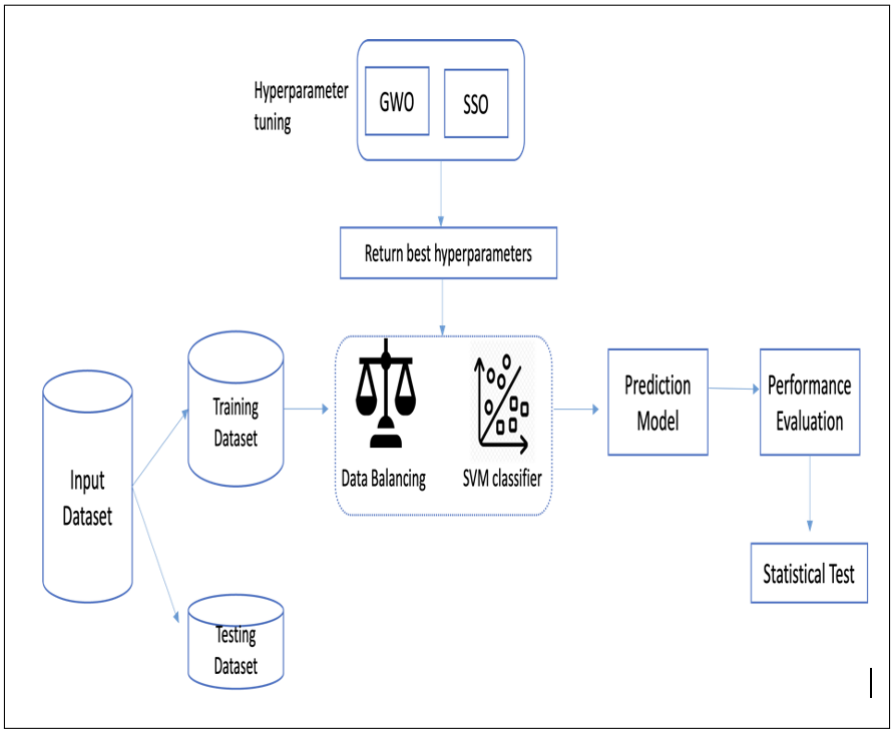

Figure 6.1: Proposed methodology

To ensure that the results were reliable and generalizable, the study adopted the widely accepted 10-fold cross-validation technique during model training and evaluation. This method helps prevent overfitting and provides a robust estimate of the model's predictive performance. The in-depth description of this method is present in Section 2.7.6 in Chapter 2, where it is defined and discussed in detail.

6.2.4 Performance Metrics

Various performance measures exist for classification prediction problems. We employ two performance evaluation measures - Area under the Receiver's Operating Curve (AUC) and F-measure - appropriate for validation of the developed model on noisy and imbalanced datasets [58, 203]. These are the most popularly used performance metrics in the SDP domain. The details about these measures are described in Section 2.7.7.

6.2.5 Statistical Evaluation

We validate the findings of this study with the help of a statistical test and ascertain whether there are any significant differences between different techniques. A non-parametric Friedman test is utilized in this study as it can be applied to independent groups, and also, the data distribution need not be normal [116]. Mean ranks are obtained amongst different techniques that are compared according to their predictive performance.

6.3 Results and Analysis

6.3.1 Result Analysis based on RQ1

RQ1. How does parameter optimization using GWO and SSO impact the performance of SVM classifier in SDP using the AEEEM datasets?

We study the influence of parameter optimization on the performance of the SVM classifier in SDP, where we constructed a predictive model using AEEEM datasets. We compare the SVM classifier performance before and after using parameter

optimization techniques – GWO and SSO. During model training, the 10-fold cross-validation method is used, and the random oversampling method is employed to balance the datasets. Table 6.1 reports the values of F-measure and AUC.

Table 6.1: F-measure and AUC scores

Datasets	**F-measure**			**AUC**		
	SVM	**GWO-SVM**	**SSO-SVM**	**SVM**	**GWO-SVM**	**SSO-SVM**
EQ	**0.747**	0.710	0.711	0.795	**0.837**	**0.837**
JDT	0.753	**0.764**	0.758	**0.858**	**0.858**	0.856
PDE	0.609	**0.629**	0.627	0.726	**0.751**	**0.751**
Lucene	0.551	**0.661**	**0.661**	0.699	**0.801**	0.800
Mylyn	0.606	**0.633**	**0.633**	**0.782**	0.769	0.768

The results, as presented in Table 6.1, demonstrate notable improvements in both F1-measure and AUC scores when SVM models are tuned using GWO and SSO. Without tuning, SVM is able to obtain the best F1-measure on the EQ (0.747) dataset only and best AUC scores on JDT (0.858) and Mylyn (0.782) datasets. However, after optimization with GWO and SSO, SVM displayed consistently high performance. SVM tuned with GWO shows an improvement in F1-score varying from 1.5% to 17% on the four datasets and an improvement in AUC scores from 3.3% to 12.8%. Similarly, SVM tuned with SSO shows an improvement in F1-score varying from 3% to 17% on four datasets and an improvement in AUC score varying from 3% to 12.6%.

In conclusion, we can say that optimizing the parameters using the GWO and SSO algorithms enhances the performance of the SVM algorithm, showing its efficacy in SDP tasks across various datasets.

6.3.2 Result Analysis based on RQ2

RQ2. What are the optimal parameter settings obtained through GWO and SSO for SVM classifiers in software defect prediction?

We derive the optimal parameter settings of the SVM algorithm by employing GWO and SSO optimization techniques and thoroughly examining the performance of SVM based on the fitness function value obtained using different population sizes and number of epochs. We have chosen the population size = 30 and maximum iterations = 100 to achieve the best performance. The fitness function for GWO and SSO is chosen to maximize the model's classification performance (AUC). Consequently, the optimal solution attained by GWO and SSO reflects the best parameters (C and γ) for SVM displayed in Table 6.2.

Table 6.2: Optimized parameters of SVM

Datasets	**GWO-SVM**		**SSO-SVM**	
	C	γ	**C**	γ
EQ	286.159089	0.00013068	387.311473	0.0001
JDT	16.4120011	0.01104883	102.177639	0.00215814
PDE	849.795898	0.00021005	933.009	0.000189
Lucene	596.5203	0.00019238	992.751747	0.0001155
Mylyn	82.8194346	0.00010074	82.3729853	0.0001

6.3.3 Result Analysis based on RQ3

RQ3. How do the optimized SVM models using GWO and SSO compare with Random search and Grid search in SDP? We have conducted a comprehensive comparison of the performance of tuned SVM models using GWO and SSO with traditional parameter tuning methods – Random Search and Grid Search– across all five AEEEM datasets. Figure 6.2 and Figure 6.3 show the F-measure and AUC scores of all the methods over the examined datasets. As seen in Figure 6.2, the highest F-measure

(0.752) value over the EQ project is produced by Random Search. For JDT, PDE, and Lucene projects, the highest F-measure values are produced by GWO-SVM. Grid Search gives the highest F-measure value on the Mylyn project. From Figure 6.3, we can observe that EQ, JDT, PDE, and Lucene have the highest AUC scores produced by GWO-SVM. For the Mylyn project, SVM without any parameter tuning gives the highest AUC (0.782).

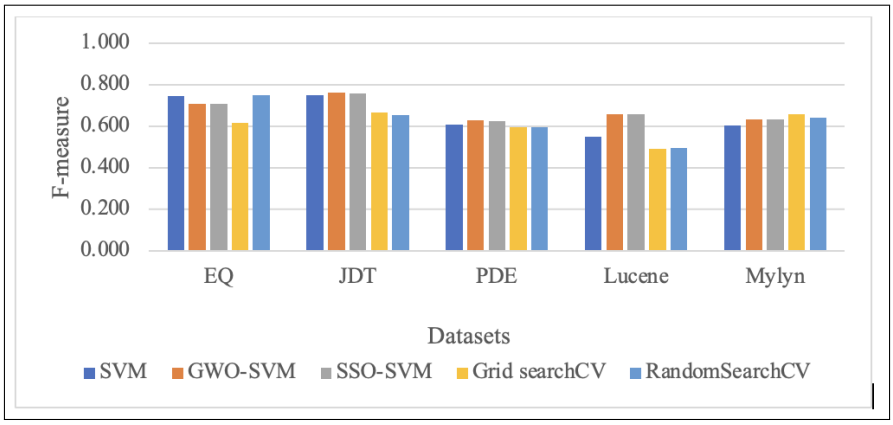

Figure 6.2: Comparative results based on F-measure

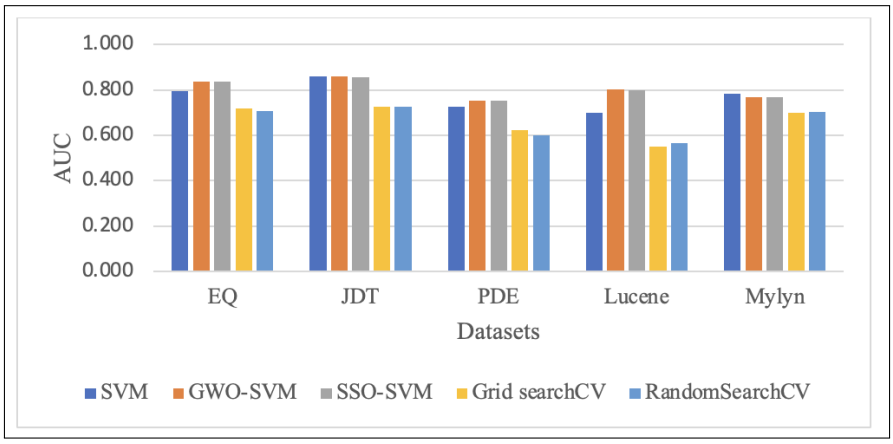

Figure 6.3: Comparative results based on AUC

For statistical validation, we conduct the Friedman Test on the F-measure and

AUC values of GWO-SVM and SSO-SVM techniques to find their mean ranks. The test was performed at a 0.05 significance level.

Table 6.3 depicts the mean ranks of all the techniques. GWO-SVM secured the highest mean rank of 3.8 and 4.5 based on F-measure and AUC values, followed by SSO-SVM with mean ranks of 3.6 and 3.8 based on F-measure and AUC values, respectively. For F-measure, the obtained p-value = 0.44 and Friedman's statistic value = 3.755. In the case of AUC, the obtained p-value = 0.003 and Friedman's statistic value = 16.289. As p-value ¡ 0.05 in the case of AUC, it implies that GWO and SSO significantly perform better than other techniques. These optimization techniques demonstrate notable improvements, highlighting their value in enhancing the efficacy of SVM classifiers for defect prediction across diverse datasets.

Table 6.3: Results of the Friedman test based on F-measure and AUC

Techniques	**Mean Rank (F1)**	**Mean Rank (AUC)**
SVM	2.8	3.7
GWO-SVM	**3.8**	**4.5**
SSO-SVM	3.6	3.8
GridSearch	2.2	1.6
RandomSearch	2.6	1.4

6.4 Discussion

This chapter explored the use of two bio-inspired metaheuristic algorithms – GWO and SSO– for tuning the hyperparameters of the SVM classifier in the context of SDP. The research was conducted on five open-source, widely studied, and imbalanced datasets from the AEEEM repository and compared with traditional SVM models as well as conventional hyperparameter tuning techniques such as Grid Search and Random Search.

The experimental findings demonstrate that parameter tuning plays a pivotal role in improving the predictive capability of SVM classifiers. Notably, both GWO-SVM and SSO-SVM models showed consistent gains in F-measure and AUC, which are particularly critical metrics in imbalanced classification tasks such as SDP. For instance, improvements in AUC ranged between 3.3% and 12.8% for GWO-SVM and between 3% and 12.6% for SSO-SVM, relative to the baseline SVM model. Similarly, F-measure gains reached up to 17%, underscoring the effectiveness of metaheuristic optimization in enhancing classifier sensitivity and precision.

Furthermore, the Friedman statistical test was applied to validate the observed improvements. The results indicated that GWO-SVM achieved the highest mean rank in both AUC and F-measure, while SSO-SVM followed closely. In the case of AUC, the p-value was found to be below the 0.05 significance threshold, confirming the statistical superiority of the optimized models over conventional approaches. This affirms that the performance enhancements achieved through GWO and SSO are not only empirically significant but also statistically reliable.

The robustness of the proposed models is also evident from their performance across diverse datasets. While traditional tuning methods such as Grid Search and Random Search occasionally produced competitive results on individual datasets, they lacked the consistency and overall effectiveness of the metaheuristic approaches. This suggests that metaheuristic algorithms are better suited for navigating complex, high-dimensional search spaces, particularly in scenarios involving imbalanced and noisy software defect data.

In addition to performance gains, the chapter sheds light on the optimal parameter ranges for the SVM's C and γ values, offering practical guidance for future applications. The chosen fitness function based on AUC effectively guided the search process, further validating the design choices within the optimization framework.

Chapter 7

Hybrid Model for SDP using parameter tuning of Ensemble classifiers

7.1 Introduction

The automated defect prediction process is implemented through ML, which is becoming one of the most powerful technologies, using historical defect data to train models for patterns that would indicate a fault [47]. Among the multiple ML methodologies, ensemble learning models, which combine the predictions from multiple base classifiers, have gained enormous attention owing to their strength and robustness in complicated datasets [78].

Despite their effectiveness, the performance of ensemble classifiers heavily depends on how well the hyperparameters are tuned. Conventional hyperparameter optimization methods such as grid search and random search are computationally

expensive and may fail to capture the complex interactions between hyperparameters [229]. There are various parameters present in each ensemble classifier that influence its learning behaviour and predictive capabilities. For instance, parameters like learning rates affect convergence patterns, the number of estimators controls model complexity, and tree depth determines how the model captures feature interactions. Research has demonstrated that optimizing these parameters can lead to substantial improvements in prediction accuracy.

This has motivated the adoption of meta-heuristic optimization algorithms, which offer a more efficient and effective alternative by mimicking natural processes such as swarm behaviour or predator-prey dynamics to explore the hyperparameter space. Meta-heuristic techniques such as GWO and SSO have demonstrated remarkable success in optimizing hyperparameters across various domains but have yet to be thoroughly explored in the context of ensemble-based SDP models. These optimization techniques have shown greater efficacy than manual parameter selection by thoroughly exploring the huge parameter space to find optimal configurations.

This study seeks to bridge this gap by exploring the potential of GWO and SSO for improving the predictive performance of ensemble classifiers-Random Forest, Gradient Boosting, AdaBoost, and XGBoost- for defect prediction. By optimizing key hyperparameters of these classifiers, we aim to achieve significant improvements in their ability to identify defect-prone modules. Additionally, we compare the effectiveness of GWO and SSO to identify which optimization algorithm yields better results in tuning ensemble models for SDP. Five open-source software projects from the AEEEM repository are used to validate the hybrid models. Before model development, we use the SMOTE technique to balance the training data because the datasets are naturally imbalanced. Additionally, we use the 10-fold cross-validation technique to mitigate overfitting during model training.

Our contributions are threefold:

1. **Evaluation of Meta-Heuristic Optimization for Ensemble Models:** We systematically evaluate the impact of hyperparameter tuning using GWO and SSO on the predictive performance of four widely used ensemble classifiers.

2. **Empirical Analysis on Real-World Datasets:** Experiments are conducted on five datasets from the AEEEM repository, employing rigorous evaluation metrics including AUC, MCC, and F-measure to assess model performance.

3. **Comparative and Statistical Analysis:** We provide a detailed comparison between GWO and SSO in terms of optimization effectiveness and analyze the statistical significance of performance improvements over non-optimized models using the Wilcoxon signed-rank test.

We propose a novel approach combining meta-heuristic optimization algorithms (GWO and SSO) with ensemble learning models to enhance SDP performance. Our study addresses three primary research questions:

- RQ1) How effective are meta-heuristic optimization algorithms (GWO and SSO) in optimizing the hyperparameters of ensemble classifiers for SDP? Which optimization algorithm (GWO or SSO) achieves better performance?

- RQ2) Which ensemble classifier shows the most significant improvement after optimization?

- RQ3) Do optimized ensemble models outperform traditional non-optimized defect prediction models?

The paper is arranged as follows: Section 2 briefly describes the methodology, including dataset collection, preprocessing steps, model development and validation steps, performance measures, and statistical tests used. Section 3 evaluates the performance of the hybrid algorithms and discusses the results. Section 4 provides the discussion of the chapter. The results of this chapter are communicated in [230].

7.2 Research Methodology

7.2.1 Dataset Collection and Variables

The study analyzed five publicly available datasets from the AEEEM repository. These datasets - *EQ*, *JDT*, *Lucene*, *Mylyn*, and *PDE* - have been extensively used in empirical software engineering research to evaluate and compare various predictive models and defect detection methodologies. The datasets have been previously discussed in detail in Section 2.7.2.

7.2.2 Data Preprocessing

Prior to model training, the input data was normalized to address the varying value ranges across different features. To achieve this, the min-max normalization method was employed, scaling all feature values within the range of $[0, 1]$. Additionally, due to the significant class imbalance present in the datasets, the training data was balanced using the SMOTE balancing approach to enhance the model's ability to generalize and reduce the likelihood of overfitting. These methods are described in detail in Section 2.7.3 and 2.7.5

7.2.3 Model Development and Validation

The proposed framework enhances Software Defect Prediction (SDP) through meta-heuristic hyperparameter optimization of ensemble learning models, as illustrated in Figure 7.1. The approach begins by splitting the input dataset into training and testing subsets, followed by applying the pre-processing step of SMOTE (Synthetic Minority Over-sampling Technique) to the training data in order to address class imbalance issues commonly found in defect datasets. Four ensemble classifiers—

Random Forest, Gradient Boosting, AdaBoost, and XGBoost—are then employed for defect prediction, with their hyperparameters optimized using two meta-heuristic algorithms: Grey Wolf Optimizer (GWO) and Salp Swarm Optimizer (SSO). These bio-inspired optimization algorithms efficiently explore the hyperparameter space to identify optimal configurations that maximize the model's performance, returning the best hyperparameter settings for each ensemble model. The optimized models are subsequently trained on the balanced training dataset and evaluated on the testing dataset using performance metrics including AUC, MCC, and F-measure. Finally, the Wilcoxon signed-rank test is conducted to statistically validate the significance of performance improvements achieved by the optimized models compared to their non-optimized counterparts, ensuring robust evaluation across five datasets.

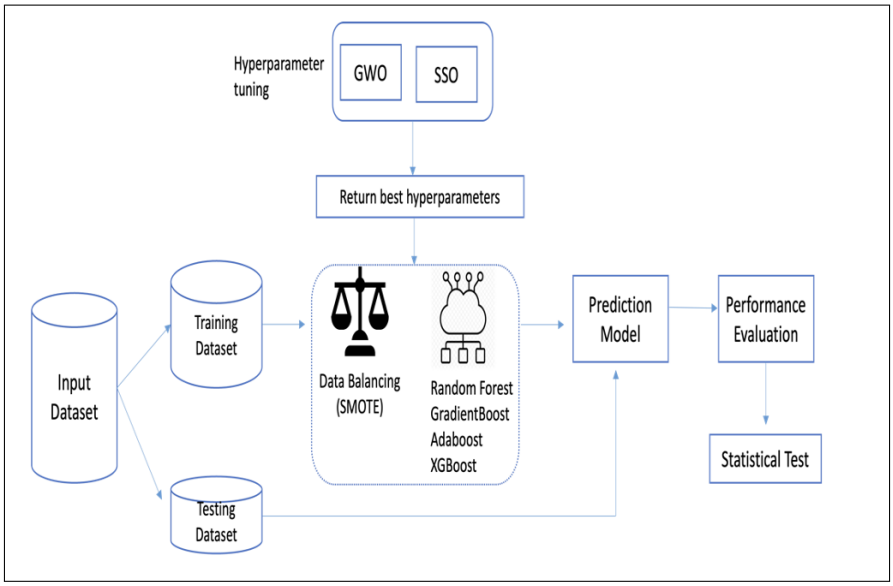

Figure 7.1: Proposed methodology

In this study, 10-fold cross-validation is employed to ensure robust and unbiased model evaluation. This technique divides the dataset into ten equal parts, using nine for training and one for testing in each iteration, ensuring that the entire dataset is

effectively utilized. Averaging results across all folds provides a reliable estimate of model performance, mitigates overfitting, and ensures efficient utilization of the entire dataset. This method is described in detail in Section 2.7.6.

7.2.4 Hyperparameter Settings

The parameter details of the tuned parameters for all the ensemble classifiers are mentioned in Table 7.1. These parameters were chosen because they greatly impact the model's complexity, learning dynamics, and ability to generalize. The range for each parameter was set empirically, guided by established practices and previous studies in SDP.

Table 7.1: Hyperparameter settings for ensemble learning models

Technique	**Parameter**	**Description**	**Range**
Random Forest	n_estimators	Total number of decision trees in the forest	(50, 200)
	max_depth	Maximum depth of each tree	(5, 50)
	min_samples_split	Minimum number of samples required to split an internal node	(2, 20)
AdaBoost	n_estimators	Number of boosting rounds or weak learners	(50, 200)
	learning_rate	Boosting learning rate	(0.01, 1.0)
Gradient Boosting	learning_rate	Boosting learning rate	(0.01, 1.0)
	n_estimators	Number of boosting trees	(50, 200)
	max_depth	Maximum depth of each tree	(3, 10)
XGBoost	learning_rate	Boosting learning rate	(0.01, 0.3)
	max_depth	Maximum depth of each tree	(3, 10)
	n_estimators	Number of boosting iterations	(50, 200)
	subsample	Subsample ratio of the training instance	(0.5, 1.0)

7.2.5 Performance Metrics

Classification models must be rigorously evaluated using well-established performance indicators to ensure the reliability and validity of their predictions. In this chapter,

three key evaluation metrics are employed: AUC, F-measure, and MCC. These metrics have been widely recognized in the SDP literature for their effectiveness, especially in handling imbalanced and noisy datasets. Together, these metrics provide a multidimensional view of model performance, capturing not only the correctness of predictions but also the trade-offs involved. Their combined use ensures that the evaluation reflects both predictive power and robustness, which is critical in defect prediction scenarios where data imbalance is common and misclassifications can lead to costly consequences. Section 2.7.7 provides comprehensive details of these metrics.

7.2.6 Statistical Evaluation

In evaluating the impact of hyperparameter tuning on ensemble classifiers for SDP, statistical testing is vital for ensuring that performance improvements are not due to random chance. In this study, the Wilcoxon signed-rank test, a non-parametric statistical method, was employed to compare the performance of ensemble models tuned using GWO and SSO. This test is well-suited for paired comparisons and is ideal for assessing the statistical significance of differences across metrics such as AUC, MCC, and F-measure.

By applying the Wilcoxon test, we objectively validated whether the observed improvements from tuning were consistent and meaningful, thus reinforcing the reliability and robustness of the experimental findings. Section 2.7.8 gives details on the Wilcoxon test.

7.3 Results Analysis

7.3.1 Results analysis based on RQ1

RQ1) How effective are meta-heuristic optimization algorithms (GWO and SSO) in optimizing the hyperparameters of ensemble classifiers for SDP? Which optimization algorithm (GWO or SSO) achieves better performance?

The study investigates how meta-heuristic optimization techniques can improve ensemble classifiers' effectiveness in predicting software defects. We conducted experiments using five datasets from the AEEEM repository, examining ensemble classifier performance with and without parameter tuning through GWO and SSO algorithms. The methodology incorporated a 10-fold cross-validation method for model training, while the SMOTE sampling technique was employed to balance the datasets. The comparative analysis of classifier performance is presented in Table 7.2, which details the AUC, MCC, and F-measure metrics across all datasets and tuning configurations.

In terms of AUC, GWO achieved consistent improvements with an average increase of 4.3% across all classifiers. SSO demonstrated slightly more variable but potentially higher improvements, with an average AUC increase of 3.8%. The MCC metric, which is particularly important for imbalanced datasets, showed even more pronounced improvements. GWO improved MCC scores by an average of 35.2%, with the most significant improvement observed in GradientBoost. SSO similarly enhanced MCC performance with an average improvement of 34.08%. The F-measure results further confirmed the effectiveness of both optimization approaches, with GWO achieving an average improvement of 6.5% and SSO showing a 6.2% increase.

The consistent improvements across all three performance metrics strongly indicate that both GWO and SSO are effective in optimizing ensemble classifiers for

SDP, with GWO showing more stable improvements and SSO achieving higher peak performances in specific cases.

Table 7.2: Average Performance Metrics Across All Datasets

Optimizer	Metric	GradientBoost	AdaBoost	XGBoost	RandomForest
No Tuning	AUC	0.7252	0.7442	0.7629	0.8317
	MCC	0.2623	0.3491	0.3074	0.4258
	F-measure	0.6752	0.6904	0.7094	0.7293
GWO	AUC	0.7946	0.7650	0.7852	0.8337
	MCC	0.3541	0.3932	0.3728	0.4384
	F-measure	0.7194	0.6996	0.7154	0.7319
SSO	AUC	0.8049	0.7565	0.7610	0.8306
	MCC	0.3517	0.3638	0.3252	0.4293
	F-measure	0.7172	0.6937	0.7120	0.7307

7.3.2 Results analysis based on RQ2

RQ2) Which ensemble classifier shows the most significant improvement after optimization?

We have conducted a comprehensive analysis for all the ensemble classifiers (GradientBoost, AdaBoost, XGBoost, RandomForest) in optimization through GWO and SSO algorithms across all performance metrics. GradientBoost demonstrates the most significant improvement after optimization among all ensemble classifiers. For the AUC metric, GradientBoost achieved the highest relative improvement with a 9.57% increase (from 0.7252 to 0.7946) using GWO and an even higher 10.99% improvement (from 0.7252 to 0.8049) using SSO. The MCC metric shows an even more dramatic enhancement for GradientBoost, with a 35.0% improvement using GWO (from 0.2623 to 0.3541) and 34.08% with SSO (from 0.2623 to 0.3517), which is particularly significant given MCC's sensitivity to class imbalance. The F-measure results further confirm GradientBoost's superior response to optimization, showing a 6.55% improvement with GWO (from 0.6752 to 0.7194) and 6.22% with

SSO (from 0.6752 to 0.7172). In contrast, other ensemble classifiers showed more modest improvements: AdaBoost demonstrated moderate gains (AUC: +1.65%, MCC: +4.21%, F-measure: +0.48% with SSO), XGBoost showed consistent but smaller improvements (AUC: +2.92%, MCC: +21.3%, F-measure: +0.85% with GWO), while RandomForest, despite having the highest baseline performance, showed minimal improvement (AUC: +0.24%, MCC: +2.96%, F-measure: +0.36% with GWO).

GradientBoost, as the ensemble classifier, emerges as the most responsive to hyperparameter optimization, and has therefore shown significant improvement among all other classifiers.

7.3.3 Results analysis based on RQ3

RQ3: Do optimized ensemble models outperform traditional non-optimized defect prediction models?

To demonstrate the performance of optimized ensemble models vs non-optimized ensemble models, we conducted a statistical test showing the significant difference between each of them. The Wilcoxon signed-rank test was performed on untuned and tuned ensemble models, considering all the performance metrics - AUC, MCC, and F-measure values. The results of the Wilcoxon signed-rank test are depicted in Table 7.3.

The analysis across all five datasets reveals that the optimized models consistently outperform their non-optimized versions in 87% of cases. The p-values obtained for no tuning vs GWO and SSO are less than 0.05 with respect to AUC, MCC, and F-measure values, suggesting a statistically significant difference between non-optimized and optimized models. The other p-values (all > 0.05) indicate no statistically significant difference between GWO and SSO. Both methods show significant improvements over no tuning, but are not significantly different from each other. These results

Table 7.3: Statistical Significance Test Results (Wilcoxon Signed-Rank Test)

Comparison	Metric	Wilcoxon p-value
No Tuning vs GWO	AUC	0.0023
No Tuning vs SSO	AUC	0.0027
GWO vs SSO	AUC	0.3844
No Tuning vs GWO	MCC	0.0018
No Tuning vs SSO	MCC	0.0022
GWO vs SSO	MCC	0.4156
No Tuning vs GWO	F-measure	0.0031
No Tuning vs SSO	F-measure	0.0035
GWO vs SSO	F-measure	0.3922

conclusively demonstrate that optimized ensemble models provide superior defect prediction capabilities compared to traditional non-optimized approaches.

7.4 Discussion

This chapter presented the development and evaluation of SDP models using four ensemble classifiers - Random Forest, Gradient Boosting, AdaBoost, and XGBoost - whose hyperparameters were tuned using two bio-inspired metaheuristic algorithms: GWO and SSO. The experimental evaluation was conducted on five widely used and publicly available software datasets from the AEEEM repository, which are inherently imbalanced in nature. To handle class imbalance and ensure robust training, the SMOTE oversampling technique was applied to the training datasets. Model validation was carried out using the ten-fold cross-validation approach, and model performance was assessed using three widely recognized and unbiased performance measures: AUC, F-measure, and MCC. These metrics are well-suited for evaluating predictive models on imbalanced and noisy datasets. The study also included statistical analysis using the Wilcoxon signed-rank test to confirm the significance of improvements

observed after hyperparameter tuning.

The main contribution of this chapter is to demonstrate that metaheuristic-based hyperparameter tuning significantly enhances the performance of ensemble classifiers in defect prediction tasks. The optimization techniques explored - GWO and SSO - proved to be effective in navigating the hyperparameter space and identifying optimal configurations that yielded considerable performance improvements.

The key findings of the chapter are reported as follows:

1. All four ensemble classifiers showed improvement in AUC, MCC, and F-measure values after being optimized using GWO and SSO. In particular, Gradient Boosting exhibited the most significant improvement, with AUC increasing by up to 10.99%, MCC by 35.0%, and F-measure by 6.55%.

2. Both GWO and SSO consistently improved the baseline (untuned) performance of all classifiers. On average, GWO demonstrated more stable gains across models, while SSO achieved higher peak performance in certain configurations.

3. The Wilcoxon signed-rank test results confirmed that the performance gains achieved through tuning were statistically significant. Optimized models outperformed their non-optimized counterparts in 87% of cases, with p-values less than 0.05 across all three metrics, thereby validating the effectiveness of hyperparameter optimization.

4. The use of metaheuristic algorithms proved to be computationally efficient and scalable for tuning ensemble classifiers. The results affirm that bio-inspired algorithms such as GWO and SSO are viable alternatives to conventional grid and random search methods, especially in high-dimensional and complex search spaces.

These findings emphasize that thoughtful hyperparameter tuning - particularly via metaheuristic algorithms - is essential in realizing the full potential of ensemble classifiers in software defect prediction. The experimental and statistical results provide strong evidence that optimized ensemble models can substantially improve defect detection accuracy, especially when dealing with imbalanced and complex real-world datasets.

Thus, the outcomes of this chapter are highly relevant for software practitioners and researchers seeking to build high-performing and reliable defect prediction models. The integration of optimization techniques not only enhances predictive performance but also contributes to more efficient and cost-effective software quality assurance processes.

Chapter 8

Conclusion

8.1 Summary of the Research Work

Software Defect Prediction (SDP) is a critical aspect of the software development lifecycle, enabling early identification of potential issues within code. As software systems grow in complexity, the likelihood of defects increases, necessitating proactive strategies to maintain quality and reliability. The primary aim of the work conducted in this thesis is to develop models that improve the prediction of defect-prone areas, facilitating efficient allocation of limited testing and maintenance resources. Accurate defect prediction allows organizations to prevent costly post-release failures and maintain high customer satisfaction. This thesis proposes and evaluates several techniques to optimize SDP models, ranging from traditional machine learning approaches to advanced metaheuristic optimization and deep learning methods. Given the promising performance of these methods in enhancing predictive accuracy, the work also explores ensemble techniques and automated hyperparameter tuning to further refine results. Through structured empirical experiments, the research presented in this thesis demonstrates the effectiveness of these approaches, providing valuable insights for

researchers and practitioners aiming to improve software quality assurance.

To comprehensively assess and evaluate the role of hybrid techniques in SDP, this study conducted a systematic literature review of 72 primary studies published between January 2000 and December 2021. The review investigated the integration of search-based and ML/DL techniques, examining their application in model development, feature selection, and parameter optimization. The analytical framework examined various research components reported across the selected primary studies, including dataset characteristics, software quality metrics employed, objective fitness functions, performance evaluation measures, validation methodologies, and statistical tests. PROMISE and NASA repositories emerged as the most widely used datasets, with traditional static code metrics serving as the primary predictors in a majority of studies. Accuracy, recall, and F-measure were the most commonly reported performance measures, and K-fold cross-validation was the predominant validation strategy. The analysis revealed that evolutionary algorithms-most notably Genetic Algorithms- and swarm intelligence approaches such as Particle Swarm Optimisation are the most frequently adopted search-based techniques, while Support Vector Machines and Neural Networks dominate the ML category. GA-SVM consistently achieved superior predictive performance across multiple datasets, outperforming most other hybrid, machine learning, and statistical techniques, as confirmed through statistical tests such as the Wilcoxon signed-rank test. Significantly, only 33% of the studies reported threats to validity, highlighting areas for methodological improvement and reliability of results. This systematic literature review identified key research gaps, such as the limited availability of industrial datasets, the underuse of multi-objective optimisation objectives, the lack of comparative validation, statistical testing across diverse datasets, and insufficient reporting of threats to validity.

High dimensionality and class imbalance are critical issues in SDP that can adversely affect the predictive model performance and generalizability. Feature

selection plays a vital role in this context because it ensures that only the most relevant and non-redundant software metrics are retained. This not only enhances classification performance but also reduces model complexity, lowers computational costs, and mitigates overfitting. Similarly, class imbalance poses difficulties during model training, frequently causing classifiers to be biased towards the majority class. To address this, appropriate data resampling techniques can be applied before the learning process.

The proposed BPSO-SMOTE-ANN model effectively demonstrated the benefits of integrating feature selection with imbalance handling. Empirical evaluation across five NASA datasets showed that the model consistently outperformed state-of-the-art ML and hybrid approaches, achieving AUC values ranging from 0.74 to 0.91 and G-mean values between 0.62 and 0.86. Statistical validation using the Friedman and Wilcoxon tests confirmed the significance of these improvements, with the model achieving the highest ranks for both performance metrics. By focusing on relevant features and eliminating redundancy, the BPSO-SMOTE-ANN model reduced training time, improved predictive capability, and concentrated on defect-prone modules, thereby lowering maintenance costs and contributing to the development of higher-quality software systems.

Given the effectiveness of hybrid algorithms in previous studies demonstrating the enhancement in SDP model performance through feature selection, we proposed a multi-filter wrapper feature selection approach. It combined three filter methods - IG, CS and RF methods with a wrapper method based on the OBWOA algorithm. The selected features were then used to train a 1D-CNN enhanced with a coordinate attention mechanism. This approach was comprehensively evaluated on 17 open-source datasets and achieved an average feature reduction of nearly 50%, effectively addressing issues of high dimensionality and redundancy while maintaining computational efficiency. The Wilcoxon signed-rank test confirmed the statistical significance of AUC im-

provements (p = 0.015). Moreover, the integration of coordinate attention within the 1D-CNN architecture outperformed traditional 2D-CNNs, yielding enhancements of up to 11% in F-measure and 35% in MCC. When compared against 18 existing ML and hybrid approachesâincluding SVM, Random Forest, PCA-ANN, BERT, and CodeBERT- the proposed model consistently achieved superior performance, securing the highest AUC in the majority of datasets. These findings underscore the importance of coupling robust feature selection strategies with deep learning architectures for SDP. The proposed hybrid framework not only demonstrated strong predictive capability but also contributed to advancing methodological research in feature selection and optimization within SDP.

In SDP, feature selection alone is not sufficient to achieve high predictive performance. Model effectiveness is also strongly influenced by hyperparameter tuning, as classifiers configured with default parameters often fail to capture the specific learning patterns required for defect-prone datasets. Properly tuned hyperparameters enhance the classifier's ability to generalize, while optimized preprocessing establishes a stronger data foundation. Together, they enable the development of robust and reliable defect prediction models. This thesis employed advanced metaheuristic optimization algorithms - GWO and SSO - in combination with machine learning and ensemble classifiers to construct more effective hybrid models for SDP. The experimental evaluation was conducted on five imbalanced datasets from the AEEEM repository. In one study, hyperparameter tuning was applied to the RBF kernel of the SVM classifier. The results demonstrated that both GWO-SVM and SSO-SVM achieved substantial improvements over baseline SVM and conventional tuning methods such as Grid Search and Random Search. Specifically, the optimized models reported AUC gains between 3â13% and F-measure improvements of up to 17%. The Friedman and Wilcoxon tests confirmed the statistical significance of these improvements. Furthermore, GWO-SVM and SSO-SVM exhibited greater robustness

and consistency across datasets compared to traditional approaches, highlighting the suitability of metaheuristic optimization for handling high-dimensional and imbalanced defect data. The study also identified practical parameter ranges for SVM's C and γ, providing useful guidance for future applications.

The application of GWO and SSO was extended to ensemble classifiers, after their proven efficacy in optimizing SVM classifier. Four ensemble methods - Random Forest, Gradient Boosting, AdaBoost, and XGBoost - were optimized using these two algorithms. Across all models, consistent improvements were observed in AUC, MCC, and F-measure. Among them, Gradient Boosting exhibited the most significant gains, with AUC increasing by 10.99%, MCC by 35%, and F-measure by 6.55%. While GWO delivered more stable improvements across classifiers, SSO achieved higher peak performance in certain configurations. Statistical validation using the Wilcoxon signed-rank test confirmed that optimized models outperformed their baselines in 87% of cases ($p < 0.05$). Collectively, these findings underscore the crucial role of hyperparameter optimization in enhancing the predictive performance of SDP models. By demonstrating the consistent effectiveness of GWO and SSO across both ML classifier and ensemble methods, this work highlights the value of metaheuristic-based tuning in building high-performing, scalable, and cost-effective defect prediction models suitable for real-world applications.

8.2 Application of the Work

The work presented in this thesis is anticipated to be of significant value to the researchers, software practitioners and society in the following ways:

Aid to Researchers:

- This work provides a strong foundation for exploring advanced hybrid mod-

els. Researchers can build upon the comparative studies of machine learning classifiers and hybrid approaches to develop novel models or optimize existing ones.

- The insights gained from this study serve as a benchmark for performance comparisons, helping researchers validate new methodologies against established metrics.

- By highlighting the impact of feature selection and hyperparameter tuning, this study encourages future research in automating optimization processes in deep learning, machine learning, and data balancing techniques for better model performance in SDP.

Aid to Software Practitioners:

- The application of optimized machine learning, deep learning techniques and feature selection strategies can be directly employed in real-world projects to enhance defect prediction, reducing the likelihood of bugs in software products.

- By focusing on selecting relevant features and tuning classifiers effectively, software engineers can reduce computational costs, execution time, and memory usage, leading to more efficient software development pipelines.

- Hybrid SDP can assist project managers in identifying potential delivery risks related to software quality. This increases the reliability of delivery commitments and stakeholder confidence.

Aid to Society:

- With improved defect prediction models, this work contributes to the development of more reliable and robust software systems, leading to fewer bugs and

crashes, in everyday applications - be it in healthcare, transportation, finance, or communication.

- As software becomes more dependable and predictable, users and businesses alike gain confidence in the systems they rely on, promoting technological adoption and innovation.
- By reducing software maintenance expenses and post-release defect corrections, thereby promoting a more efficient and sustainable software development ecosystem.

8.3 Future Work

While this thesis explores a diverse range of hybrid algorithms integrating search-based and ML/DL techniques for SDP, these studies could be expanded in the future to datasets from different application areas and software projects created in different programming languages. The generalizability of the results and practical relevance across various software ecosystems would be improved by extending the evaluation to a broader selection of industrial and real-world systems.

Further research could potentially explore the application of advanced evolutionary and swarm intelligence algorithms to design more adaptive hybrid models. Recent metaheuristic algorithms that have been developed in the past few years, such as the Reptile Search algorithm, Marine Predators Algorithm (MPA), Aquila Optimizer (AO) or their hybridized versions of existing methods, could be applied to improve feature selection, parameter tuning, and model optimization. Evaluation of these algorithms on imbalanced and large-scale datasets would contribute to developing more robust and scalable SDP techniques.

Hyperparameter optimization can be expanded beyond classifiers to incorporate

additional preprocessing methods like feature extraction, data normalization, and resampling techniques may be beneficial for future research. Using hybrid techniques to optimize these preprocessing steps could increase model accuracy, lower computing overhead, and improve adaptability to various software projects.

Moreover, multi-objective optimization should be considered to find a balance between goals such as predictive accuracy, computational cost, interpretability, and scalability. Applying hybrid metaheuristic approaches capable of handling multiple objectives simultaneously can result in more adaptable and realistic models that are appropriate for industrial applications.

Replication and reproducibility are necessary to establish a more robust evidence base in SDP. It is recommended that future research should replicate the experiments reported in this thesis on new datasets and domains so that results are validated across different real-world environments. There should be transparency in the reporting of experimental setups, parameter values, and threats to validity, which will also facilitate researchers and practitioners in evaluating the transferability of findings better.

Finally, with the quick advancements in deep learning and large-scale automation, future research could examine integrating metaheuristic optimization with advanced neural architectures such as Transformers, Graph Neural Networks, or hybrid CNN-RNN frameworks. These models, when paired with automated optimization pipelines, could push the boundaries of defect prediction toward higher predictive accuracy, greater interpretability, and wider industrial adoption.

Appendix

Hybridized techniques used in primary studies

Table A.1 lists all the search-based and machine learning algorithms employed in the hybridized techniques for developing software defect prediction (SDP) models.

Table A.1: Algorithms used in primary studies

Study #	Algorithms Used
HS1	MOGP-DT
HS2	ES/HS/AHS-NB/MLP/SVM/LR/KNN
HS3	ES-PNN, SA-PNN
HS4	PSO-Integrated Decision Network
HS6	Evolutionary Programming with Least Square SVM
HS7	PSO-Bagging-LDA/LR/NB/KNN/K*/BPNN/SVM/LibSVM/CART/C4.5/Random Forest
HS11	GA/PSO-Bagging-LR/LDA/NB/KNN/K*/BPNN/SVM/C4.5/CART/RF
HS13	ABC-ANN
HS14	Quantum PSO-ANN
HS15	Adaptive PSO-ANN
HS16	GA-Ensemble Learning
HS17	Harmony Search-NB/LR/DT
HS18	GP-NB, GP-C4.5
HS20	MOGA-NSGA-II
HS21	CRBA-SVM
HS23	GA-LR
HS25	Glowworm Swarm Optimization-GA-MLPNN

Table A.1 (continued): Algorithms used in primary studies

Study #	Algorithms Used
HS26	BA-RF
HS27	PSO-GA-C4.5
HS30	Firefly-KNN
HS31	MOCS-SVM
HS32	Bird Mating Algorithm-ANN
HS34	LRNN-BGA-BPSO-BACO
HS35	MOCSTSVM
HS36	GFS-MaxLB, PSO-LDA
HS37	Adaptive Two SVM-MOCS
HS38	Bound PSO-DNN
HS39	Neural network evolutionary programming, Hybrid fuzzy GBML, ADI, INT
HS40	CamNN, PSO_ACO-C, BNGE_C, Linear LMS-C
HS41	Firefly-ANN
HS43	DE-CART/RF, SA-CART/RF
HS44	Fuzzy Mutual Information ACO
HS45	Weighted PSO-C4.5, Weighted PSO-NB
HS46	HMOCSSVM
HS47	GA-DNN
HS48	MFO-KNN/DT/LDA
HS49	MOES-J48, MOES-NB, MOES-KNN, GA-J48
HS50	GA-PSO-Feedforward Neural Network/RNN/ANN/DNN
HS51	MOFES-NSGA-II, MOFES-MO-Cell, MOFES-SPEA2, MOFES-PAES, MOFES-SMSEMOA
HS52	GA/PSO-Stacking-MLP-DT/RF/SVM/KNN/BN

Table A.1 (continued): Algorithms used in primary studies

Study #	Algorithms Used
HS53	PSO-ELM, PSO-KELM, Firefly-ELM, Firefly-KELM
HS54	GA-PSO-SVM
HS55	Binary Harris Hawk Optimization-KNN/DT/LDA
HS57	KAEA
HS58	WO-SVM, GA-WO-SVM
HS59	Firefly-RF/SVM, WSA-RF/SVM
HS60	GA-Bagging-KNN/LR/SVM/NB/DT
HS61	PSO-Modified GA-Bagging-KNN/RF/SVM/LSTSVM/MW-LSTSVM
HS62	DE-LR
HS63	ACO/ABC/BA/CS/Elephant Search/Flower Search/NSGA-II/PSO/Rhinoceros Search-NB
HS64	Multi-Objective BA-SVM
HS65	WOA-KNN/DT
HS66	SSO-BPNN
HS67	PSO-SMOTE-SVM
HS68	WOA-SA-DNN-KELM
HS69	NSGA-III-CNN-SVM
HS70	Harmony Search-Bagging-DT
HS71	Improved QPSO-KTSVM
HS72	PSO-WPA-DNN
HS12, HS24	GA-Fuzzy Logic
HS49, HS22	GA-KNN
HS49, HS63	GA-NB
HS39, HS36	HIDER
HS29, HS32, HS39, HS41	GA-ANN

Table A.1 (continued): Algorithms used in primary studies

Study #	Algorithms Used
HS39, HS42	DT-GA
HS36, HS56	GFS-LB
HS36, HS56, HS40	GFS-AB
HS8, HS54	PSO-SVM
HS58, HS5, HS9, HS23, HS54	GA-SVM
HS41, HS10	PSO-ANN
HS30, HS33, HS59	Firefly-SVM
HS30, HS63	Firefly-NB
HS53, HS23	GA-ELM
HS28, HS19	GA-Nearest Neighbor Filter

INT: Genetic algorithm-based classifier with intervalar rules; HIDER: Hierarchical decision rules; ADI: Genetic algorithm-based classifier; XCS: X classifier system; UCS: Supervised classifier system; MOGP: Multi-Objective Genetic Programming; SHO: Spotted Hyena Optimizer; AIRS: Artificial Immune Recognition System; GFS-LB: Genetic Fuzzy System-Logitboost; GFS-AB: Genetic Fuzzy System-Adaboost; GFS-MaxLB: Genetic Fuzzy SystemâLogitboost with Single Winner Inference; PSOLDA: PSO Linear Discriminant Analysis; HMOCSSVM: Hybrid multiobjective cuckoo algorithm with SVM; CRBA: Changing Range Bat Algorithm; KAEA: Kernel PCA-Adaptive Genetic AlgorithmâELMâAdaboost; MOFES: Multi-objective Feature Selection; CamNN: Nearest neighbor classification using cam weighted distance; BNGE_C: Batch Nested Generalized Exemplar Classifier; LinearLMS: Least Mean Square Linear Classifier; BACO: Binary Ant Colony Optimization; BPSO: Binary Particle Swarm Optimization; LRNN: Layered Recurrent Neural Network; BGA: Binary Genetic Algorithm; MOES: Multi-Objective Evolutionary Search; MOCS: Multi-Objective Cuckoo Search; GSO: Group Search Optimization; MOGA: Multi-objective Genetic Algorithm; MFO: Moth Flame Optimization; MLPNN: Multilayer Perceptron Neural Network; GBML: Genetics Based Machine Learning; WPA: Wolf Pack Algorithm.

Highly cited journals/conferences:

Table A.2 presents the list of the top 15 highly cited journals and conferences.

Table A.2: List of top 15 highly cited journals/conferences

Study #	Year	Type	Journal/Conference Name	Title	No. of Citations
HS2	2010	Journal	Software: Practice and Experience	Choosing software metrics for defect prediction: an investigation on feature selection techniques	266
HS16	2016	Journal	IEEE Transactions on Software Engineering	HYDRA: Massively Compositional Model for Cross-Project Defect Prediction	175
HS13	2015	Journal	Applied Soft Computing	Software defect prediction using cost-sensitive neural network	142
HS46	2019	Journal	Concurrency and Computation: Practice and Experience	An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search	104
HS1	2003	Conference	Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence	Genetic Programming-Based Decision Trees for Software Quality Classification	60
HS47	2019	Journal	Cluster Computing	Deep neural network-based hybrid approach for software defect prediction using software metrics	58
HS14	2015	Journal	Applied Soft Computing	Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization	56
HS17	2016	Journal	Applied Soft Computing	Effective multi-objective naÃ¯ve Bayes learning for cross-project defect prediction	56
HS28	2017	Journal	Information and Software Technology	A benchmark study on the effectiveness of search-based data selection and feature selection for cross-project defect prediction	55
HS7	2013	Journal	International Journal of Software Engineering and Its Applications	Combining Particle Swarm Optimization based Feature Selection and Bagging Technique for Software Defect Prediction	52
HS5	2012	Conference	SAC â12: Proceedings of the 27th Annual ACM Symposium on Applied Computing	A Further Analysis on the Use of Genetic Algorithm to Configure SVM for Inter-Release Fault Prediction	45
HS11	2014	Journal	Journal of Software	Metaheuristic Optimization based Feature Selection for Software Defect Prediction	39
HS34	2018	Journal	Expert Systems with Applications	Iterated feature selection algorithms with layered recurrent neural network for software fault prediction	39
HS3	2010	Journal	Engineering Applications of Artificial Intelligence	Exhaustive and heuristic search approaches for learning a software defect prediction model	38
HS6	2012	Journal	Information Sciences	An evolutionary programming based asymmetric weighted least squares support vector machine ensemble learning methodology for software repository mining	35

Bibliography

[1] M. D' Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, pp. 531–577, 2012.

[2] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175, 2018.

[3] M. Harman, P. McMinn, J. T. de Souza, and S. Yoo, *Search Based Software Engineering: Techniques, Taxonomy, Tutorial*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–59.

[4] M. Harman and J. Clark, "Metrics are fitness functions too," in *Proceedings of the Software Metrics, 10th International Symposium*. IEEE Computer Society, 2004, p. 58â69.

[5] Y. Singh and R. Malhotra, *Object-Oriented Software Engineering*. PHI Learning Pvt. Ltd., 2012.

[6] R. Malhotra, *Empirical research in software engineering: concepts, analysis, and applications*. CRC press, 2016.

[7] M. McDonald, R. Musson, and R. Smith, *The practical guide to defect prevention*. Microsoft Press, 2007.

[8] B. Boehm and V. R. Basili, "Software defect reduction top 10 list," *Computer*, vol. 34, no. 1, pp. 135–137, 2001.

[9] B. Grishma and C. Anjali, "Software root cause prediction using clustering techniques: A review," in *2015 Global Conference on Communication Technologies (GCCT)*. IEEE, 2015, pp. 511–515.

[10] Y. Jiang, B. Cukic, and Y. Ma, "Techniques for evaluating fault prediction models," *Empirical Software Engineering*, vol. 13, pp. 561–595, 2008.

[11] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," *Applied Soft Computing*, vol. 21, pp. 286–297, 2014.

[12] Z. Xu, J. Liu, X. Luo, Z. Yang, Y. Zhang, P. Yuan, Y. Tang, and T. Zhang, "Software defect prediction based on kernel pca and weighted extreme learning machine," *Information and Software Technology*, vol. 106, pp. 182–200, 2019.

[13] M. Harman, S. A. Mansouri, and Y. Zhang, *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, pp. 1–61, 2012.

[14] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[15] R. Malhotra, M. Khanna, and R. R. Raje, "On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions," *Swarm and Evolutionary Computation*, vol. 32, pp. 85–109, 2017.

[16] T. M. Khoshgoftaar, N. Seliya, and Y. Liu, "Genetic programming-based decision trees for software quality classification," in *Proceedings. 15th IEEE*

International Conference on Tools with Artificial Intelligence. IEEE, 2003, pp. 374–383.

[17] C. Grosan and A. Abraham, "Hybrid evolutionary algorithms: methodologies, architectures, and reviews," in *Hybrid Evolutionary Algorithms*. Springer, 2007, pp. 1–17.

[18] C. Jin and S.-W. Jin, "Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization," *Applied Soft Computing*, vol. 35, pp. 717–725, 2015.

[19] M. Prasad, L. Florence, and A. Arya, "A study on software metrics based software defect prediction using data mining and machine learning techniques," *International Journal of Database Theory and Application*, vol. 8, no. 3, pp. 179–190, 2015.

[20] B. Turhan and A. Bener, "Analysis of naive bayesâ assumptions on software fault data: An empirical study," *Data & Knowledge Engineering*, vol. 68, no. 2, pp. 278–290, 2009.

[21] T. M. Khoshgoftaar and J. C. Munson, "The lines of code metric as a predictor of program faults: A critical analysis," in *Proceedings Fourteenth Annual International Computer Software and Applications Conference*. IEEE Computer Society, 1990, pp. 408–409.

[22] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2006.

[23] Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect-

proneness prediction framework," *IEEE Transactions on Software Engineering*, vol. 37, no. 3, pp. 356–370, 2010.

[24] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, pp. 531–577, 2012.

[25] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[26] M. Cetiner and O. K. Sahingoz, "A comparative analysis for machine learning based software defect prediction systems," in *2020 11th International conference on computing, communication and networking technologies (ICCCNT)*. IEEE, 2020, pp. 1–7.

[27] M. Samir, M. El-Ramly, and A. Kamel, "Investigating the use of deep neural networks for software defect prediction," in *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*. IEEE, 2019, pp. 1–6.

[28] M. M. Askari and V. K. Bardsiri, "Software defect prediction using a high performance neural network," *International Journal of Software Engineering and Its Applications*, vol. 8, no. 12, pp. 177–188, 2014.

[29] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415–1425, 1989.

[30] E. Erturk and E. A. Sezer, "A comparison of some soft computing methods for software fault prediction," *Expert Systems with Applications*, vol. 42, no. 4, pp. 1872–1879, 2015.

[31] M. J. Hernández-Molinos, A. J. Sánchez-García, R. E. Barrientos-Martínez, J. C. Pérez-Arriaga, and J. O. Ocharán-Hernández, "Software defect prediction with bayesian approaches," *Mathematics*, vol. 11, no. 11, p. 2524, 2023.

[32] S. Amasaki, Y. Takagi, O. Mizuno, and T. Kikuno, "A bayesian belief network for assessing the likelihood of fault content," in *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.* IEEE, 2003, pp. 215–226.

[33] P. Reena and R. Binu, "Software defect prediction system–decision tree algorithm with two level data pre-processing," *International Journal of Engineering Research & Technology (IJERT)*, vol. 3, no. 3, 2014.

[34] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on software engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[35] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and methods of system dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.

[36] P. Singh and S. Verma, "Empirical investigation of fault prediction capability of object oriented metrics of open source software," in *2012 Ninth International Conference on Computer Science and Software Engineering (JCSSE)*. IEEE, 2012, pp. 323–327.

[37] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 12, pp. 1253–1269, 2018.

[38] F. Rahman and P. Devanbu, "How, and why, process metrics are better," in *2013*

35th international conference on software engineering (ICSE). IEEE, 2013, pp. 432–441.

[39] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *Proceedings of the 27th international conference on Software engineering*, 2005, pp. 284–292.

[40] A. E. Hassan, "Predicting faults using the complexity of code changes," in *2009 IEEE 31st international conference on software engineering*. IEEE, 2009, pp. 78–88.

[41] C. Bird, N. Nagappan, B. Murphy, H. Gall, and P. Devanbu, "Don't touch my code! examining the effects of ownership on software quality," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 4–14.

[42] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," in *2010 IEEE International Conference on Software Maintenance*. IEEE, 2010, pp. 1–10.

[43] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, no. 4, pp. 308–320, 1976.

[44] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on software engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[45] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, 2003.

[46] Y. Zhou and H. Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771–789, 2006.

[47] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2011.

[48] K. O. Elish and M. O. Elish, "Predicting defect-prone software modules using support vector machines," *Journal of Systems and Software*, vol. 81, no. 5, pp. 649–660, 2008.

[49] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[50] R. Malhotra and Y. Singh, "On the applicability of machine learning techniques for object oriented software fault prediction," *Software Engineering: An International Journal*, vol. 1, no. 1, pp. 24–37, 2011.

[51] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Computing*, vol. 22, pp. 77–88, 2019.

[52] M. Harman, "The current state and future of search based software engineering," in *Future of Software Engineering (FOSE'07)*. IEEE, 2007, pp. 342–357.

[53] C. Manjula and L. Florence, "Hybrid approach for software defect prediction

using machine learning with optimization technique," *International Journal of Computer and Information Engineering*, vol. 12, no. 1, pp. 28–32, 2018.

[54] R. S. Wahono, N. Suryana, and S. Ahmad, "Metaheuristic optimization based feature selection for software defect prediction." *J. Softw.*, vol. 9, no. 5, pp. 1324–1333, 2014.

[55] Y. Hassouneh, H. Turabieh, T. Thaher, I. Tumar, H. Chantar, and J. Too, "Boosted whale optimization algorithm with natural selection operators for software fault prediction," *IEEE Access*, vol. 9, pp. 14 239–14 258, 2021.

[56] H. Can, X. Jianchun, Z. Ruide, L. Juelong, Y. Qiliang, and X. Liqiang, "A new model for software defect prediction using particle swarm optimization and support vector machine," in *2013 25th Chinese control and decision conference (CCDC)*. IEEE, 2013, pp. 4106–4110.

[57] X. Yang, H. Yu, G. Fan, and K. Yang, "A differential evolution-based approach for effort-aware just-in-time software defect prediction," in *Proceedings of the 1st ACM SIGSOFT international workshop on representation learning for software engineering and program languages*, 2020, pp. 13–16.

[58] A. B. De Carvalho, A. Pozo, and S. R. Vergilio, "A symbolic fault-prediction model based on multiobjective particle swarm optimization," *Journal of Systems and Software*, vol. 83, no. 5, pp. 868–882, 2010.

[59] Y. Singh, A. Kaur, and R. Malhotra, "Prediction of software quality model using gene expression programming," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2009, pp. 43–58.

[60] R. Malhotra, "An extensive analysis of search-based techniques for predicting

defective classes," *Computers & Electrical Engineering*, vol. 71, pp. 611–626, 2018.

[61] R. Malhotra and K. Khan, "A novel software defect prediction model using two-phase grey wolf optimisation for feature selection," *Cluster Computing*, vol. 27, no. 9, pp. 12 185–12 207, 2024.

[62] K. Zhu, S. Ying, N. Zhang, and D. Zhu, "Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network," *Journal of Systems and Software*, vol. 180, p. 111026, 2021.

[63] H. Turabieh, M. Mafarja, and X. Li, "Iterated feature selection algorithms with layered recurrent neural network for software fault prediction," *Expert Systems with Applications*, vol. 122, pp. 27–42, 2019.

[64] F. Li, X. Rong, and Z. Cui, "A hybrid crba-svm model for software defect prediction," *International Journal of Wireless and Mobile Computing*, vol. 10, no. 2, pp. 191–196, 2016.

[65] S. Kassaymeh, S. Abdullah, M. A. Al-Betar, and M. Alweshah, "Salp swarm optimizer for modeling the software fault prediction problem," *Journal of King Saud University-Computer and Information Sciences*, vol. 34, no. 6, pp. 3365–3378, 2022.

[66] B. A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-based software engineering and systematic reviews*. CRC press, 2015.

[67] H. Alsghaier and M. Akour, "Software fault prediction using whale algorithm with genetics algorithm," *Software: Practice and Experience*, vol. 51, no. 5, pp. 1121–1146, 2021.

[68] X. Cai, Y. Niu, S. Geng, J. Zhang, Z. Cui, J. Li, and J. Chen, "An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 5, p. e5478, 2020.

[69] A. B. Nassif, M. A. Talib, M. Azzeh, S. Alzaabi, R. Khanfar, R. Kharsa, and L. Angelis, "Software defect prediction using learning to rank approach," *Scientific Reports*, vol. 13, no. 1, p. 18885, 2023.

[70] N. S. Harzevili and S. H. Alizadeh, "Analysis and modeling conditional mutual dependency of metrics in software defect prediction using latent variables," *Neurocomputing*, vol. 460, pp. 309–330, 2021.

[71] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Using the support vector machine as a classification method for software defect prediction with static code metrics," in *International Conference on Engineering Applications of Neural Networks*. Springer, 2009, pp. 223–234.

[72] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, pp. 1–27, 2011.

[73] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[74] M. A. Khan, N. S. Elmitwally, S. Abbas, S. Aftab, M. Ahmad, M. Fayaz, and F. Khan, "Software defect prediction using artificial neural networks: A systematic literature review," *Scientific Programming*, vol. 2022, no. 1, p. 2117339, 2022.

[75] I. Gondra, “Applying machine learning to software fault-proneness prediction,” *Journal of Systems and Software*, vol. 81, no. 2, pp. 186–195, 2008.

[76] S. Haykin, *Neural networks and learning machines, 3/E*. Pearson Education India, 2009.

[77] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 2002.

[78] T. G. Dietterich, “Ensemble methods in machine learning,” in *International workshop on multiple classifier systems*. Springer, 2000, pp. 1–15.

[79] B. Ghojogh and M. Crowley, “The theory behind overfitting, cross validation, regularization, bagging, and boosting: tutorial,” *arXiv preprint arXiv:1905.12787*, 2019.

[80] N. S. Thomas and S. Kaliraj, “An improved and optimized random forest based approach to predict the software faults,” *SN Computer Science*, vol. 5, no. 5, p. 530, 2024.

[81] M. J. Siers and M. Z. Islam, “Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem,” *Information Systems*, vol. 51, pp. 62–71, 2015.

[82] J. H. Friedman, “Greedy function approximation: a gradient boosting machine,” *Annals of statistics*, pp. 1189–1232, 2001.

[83] Y. Freund and R. E. Schapire, “A decision-theoretic generalization of on-line learning and an application to boosting,” *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[84] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[85] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-international conference on neural networks*, vol. 4. ieee, 1995, pp. 1942–1948.

[86] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation*, vol. 5. ieee, 1997, pp. 4104–4108.

[87] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[88] E. Emary, H. M. Zawbaa, and A. E. Hassanien, "Binary grey wolf optimization approaches for feature selection," *Neurocomputing*, vol. 172, pp. 371–381, 2016.

[89] M. Mafarja, R. Jarrar, S. Ahmad, and A. A. Abusnaina, "Feature selection using binary particle swarm optimization with time varying inertia weight strategies," in *Proceedings of the 2nd international conference on future networks and distributed systems*, 2018, pp. 1–9.

[90] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46–61, 2014.

[91] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016.

[92] M. Mafarja and S. Mirjalili, "Whale optimization approaches for wrapper feature selection," *Applied Soft Computing*, vol. 62, pp. 441–453, 2018.

[93] H. R. Tizhoosh, "Opposition-based learning: a new scheme for machine intelligence," in *International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'06)*, vol. 1. IEEE, 2005, pp. 695–701.

[94] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition versus randomness in soft computing techniques," *Applied Soft Computing*, vol. 8, no. 2, pp. 906–918, 2008.

[95] H. Wang, Z. Wu, S. Rahnamayan, Y. Liu, and M. Ventresca, "Enhancing particle swarm optimization using generalized opposition-based learning," *Information Sciences*, vol. 181, no. 20, pp. 4699–4714, 2011.

[96] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition-based differential evolution," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 64–79, 2008.

[97] R. A. Ibrahim, M. Abd Elaziz, and S. Lu, "Chaotic opposition-based grey-wolf optimization algorithm based on differential evolution and disruption operator for global optimization," *Expert Systems with Applications*, vol. 108, pp. 1–27, 2018.

[98] A. R. Malisia and H. R. Tizhoosh, "Applying opposition-based ideas to the ant colony system," in *2007 IEEE swarm intelligence symposium*. IEEE, 2007, pp. 182–189.

[99] S. Mirjalili, A. H. Gandomi, S. Z. Mirjalili, S. Saremi, H. Faris, and S. M. Mirjalili, "Salp swarm algorithm: A bio-inspired optimizer for engineering design problems," *Advances in Engineering Software*, vol. 114, pp. 163–191, 2017.

[100] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2011.

[101] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: a benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, pp. 531–577, 2012.

[102] ——, "An extensive comparison of bug prediction approaches," in *2010 7th IEEE working conference on mining software repositories (MSR 2010)*. IEEE, 2010, pp. 31–41.

[103] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th international conference on predictive models in software engineering*, 2010, pp. 1–10.

[104] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[105] S. B. Kotsiantis, D. Kanellopoulos, and P. E. Pintelas, "Data preprocessing for supervised leaning," *International Journal of Computer Science*, vol. 1, no. 2, pp. 111–117, 2006.

[106] M.-L. Zhang and Z.-H. Zhou, "Ml-knn: A lazy learning approach to multi-label learning," *Pattern Recognition*, vol. 40, no. 7, pp. 2038–2048, 2007.

[107] L. E. A. d. S. Santana and A. M. de Paula Canuto, "Filter-based optimization techniques for selection of feature subsets in ensemble systems," *Expert Systems with Applications*, vol. 41, no. 4, pp. 1622–1631, 2014.

[108] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," in *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*. IEEE, 2016, pp. 309–320.

[109] H. Liu and R. Setiono, "Chi2: Feature selection and discretization of numeric attributes," in *Proceedings of 7th IEEE international conference on tools with artificial intelligence*. Ieee, 1995, pp. 388–391.

[110] I. Kononenko, "Estimating attributes: Analysis and extensions of relief," in *European Conference on Machine Learning*. Springer, 1994, pp. 171–182.

[111] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & electrical engineering*, vol. 40, no. 1, pp. 16–28, 2014.

[112] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[113] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[114] A. P. Bradley, "Roc curve equivalence using the kolmogorov–smirnov test," *Pattern Recognition Letters*, vol. 34, no. 5, pp. 470–475, 2013.

[115] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. Jan, pp. 1–30, 2006.

[116] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The annals of mathematical statistics*, vol. 11, no. 1, pp. 86–92, 1940.

[117] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics: Methodology and distribution*. Springer, 1992, pp. 196–202.

[118] R. Malhotra, S. Chawla, and A. Sharma, "Software defect prediction using hybrid techniques: a systematic literature review." *Soft Computing-A Fusion of Foundations, Methodologies & Applications*, vol. 27, no. 12, 2023.

[119] B. Kitchenham, S. Charters *et al.*, "Guidelines for performing systematic literature reviews in software engineering," 2007.

[120] R. Malhotra, "Search based techniques for software fault prediction: current trends and future directions," in *Proceedings of the 7th International Workshop on Search-Based Software Testing*, 2014, pp. 35–36.

[121] R. Malhotra, M. Khanna, and R. R. Raje, "On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions," *Swarm and Evolutionary Computation*, vol. 32, pp. 85–109, 2017.

[122] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Machine learning based methods for software fault prediction: A survey," *Expert Systems with Applications*, vol. 172, p. 114595, 2021.

[123] L. Chen, M. A. Babar, and H. Zhang, "Towards an evidence-based understanding of electronic data sources," in *14th International conference on evaluation and assessment in software engineering (EASE)*. BCS Learning & Development, 2010.

[124] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014, pp. 1–10.

[125] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Information and software technology*, vol. 54, no. 1, pp. 41–59, 2012.

[126] T. M. Khoshgoftaar, N. Seliya, and Y. Liu, "Genetic programming-based decision trees for software quality classification," in *Proceedings. 15th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, 2003, pp. 374–383.

[127] Y. Niu, Z. Tian, M. Zhang, X. Cai, and J. Li, "Adaptive two-svm multi-objective cuckoo search algorithm for software defect prediction," *International Journal of Computing Science and Mathematics*, vol. 9, no. 6, pp. 547–554, 2018.

[128] K. Gao, T. M. Khoshgoftaar, H. Wang, and N. Seliya, "Choosing software metrics for defect prediction: an investigation on feature selection techniques," *Software: Practice and Experience*, vol. 41, no. 5, pp. 579–606, 2011.

[129] W. Geng, "Retracted: Cognitive deep neural networks prediction method for software fault tendency module based on bound particle swarm optimization," 2018.

[130] P. C. Pendharkar, "Exhaustive and heuristic search approaches for learning a software defect prediction model," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 1, pp. 34–40, 2010.

[131] M. Khari and P. Kumar, "Evolutionary computation-based techniques over multiple data sets: an empirical assessment," *Arabian Journal for Science and Engineering*, vol. 43, no. 8, pp. 3875–3885, 2018.

[132] N.-H. Chiu, "Combining techniques for software quality classification: an integrated decision network approach," *Expert Systems with Applications*, vol. 38, no. 4, pp. 4618–4625, 2011.

[133] W. Rhmann, "Cross project defect prediction using hybrid search based algorithms," *International Journal of Information Technology*, vol. 12, no. 2, pp. 531–538, 2020.

[134] F. Sarro, S. Di Martino, F. Ferrucci, and C. Gravino, "A further analysis on the use of genetic algorithm to configure support vector machines for inter-release fault prediction," in *Proceedings of the 27th annual ACM symposium on applied computing*, 2012, pp. 1215–1220.

[135] I. Arora and A. Saha, "Software fault prediction using firefly algorithm," *International Journal of Intelligent Engineering Informatics*, vol. 6, no. 3-4, pp. 356–377, 2018.

[136] L. Yu, "An evolutionary programming based asymmetric weighted least squares support vector machine ensemble learning methodology for software repository mining," *Information Sciences*, vol. 191, pp. 31–46, 2012.

[137] R. S. Wahono and N. Suryana, "Combining particle swarm optimization based feature selection and bagging technique for software defect prediction," *International Journal of Software Engineering and Its Applications*, vol. 7, no. 5, pp. 153–166, 2013.

[138] R. Malhotra, A. Rajpal, and D. Rathore, "Parameter tuning on software defect prediction using differential evolution & simulated annealing," in *2018 International Conference on Big Data and Artificial Intelligence (BDAI)*. IEEE, 2018, pp. 97–106.

[139] G. Manivasagam and R. Gunasundari, "An optimized feature selection using fuzzy mutual information based ant colony optimization for software defect prediction," *International Journal of Engineering & Technology*, vol. 7, no. 1.1, pp. 456–460, 2018.

[140] B. Shuai, H. Li, M. Li, Q. Zhang, and C. Tang, "Software defect prediction using dynamic support vector machine," in *2013 Ninth International Conference on Computational Intelligence and Security*. IEEE, 2013, pp. 260–263.

[141] L. Brezočnik and V. Podgorelec, "Applying weighted particle swarm optimization to imbalanced data in software defect prediction," in *International Conference âNew Technologies, Development and Applicationsâ*. Springer, 2018, pp. 289–296.

[142] N. Kayarvizhy, S. Kanmani, and R. Uthariaraj, "Improving fault prediction using ann-pso in object oriented systems," *International Journal of Computer Applications*, vol. 73, no. 3, pp. 0975–8887, 2013.

[143] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Computing*, vol. 22, no. Suppl 4, pp. 9847–9863, 2019.

[144] K. Li, C. Chen, W. Liu, X. Fang, and Q. Lu, "Software defect prediction using fuzzy integral fusion based on ga-fm," *Wuhan University Journal of Natural Sciences*, vol. 19, no. 5, pp. 405–408, 2014.

[145] I. Tumar, Y. Hassouneh, H. Turabieh, and T. Thaher, "Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction," *Ieee Access*, vol. 8, pp. 8041–8055, 2020.

[146] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Applied Soft Computing*, vol. 33, pp. 263–277, 2015.

[147] A. Ekundayo *et al.*, "Wrapper feature selection based heterogeneous classifiers for software defect prediction," *Adeleke University Journal of Engineering and Technology*, vol. 2, no. 1, pp. 1–11, 2019.

[148] S. I. Ayon, "Neural network based software defect prediction using genetic algorithm and particle swarm optimization," in *2019 1st International conference on advances in science, engineering and robotics technology (ICASERT)*. IEEE, 2019, pp. 1–4.

[149] B. Dhanalaxmi, G. A. Naidu, and K. Anuradha, "Adaptive pso based association rule mining technique for software defect classification using ann," *Procedia Computer Science*, vol. 46, pp. 432–442, 2015.

[150] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on pareto based multi-objective feature selection for software defect prediction," *Journal of Systems and Software*, vol. 152, pp. 215–238, 2019.

[151] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on Software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.

[152] F. Matloob, S. Aftab, and A. Iqbal, "A framework for software defect prediction using feature selection and ensemble learning techniques." *International Journal of Modern Education & Computer Science*, vol. 11, 2019.

[153] D. Ryu and J. Baik, "Effective multi-objective naïve bayes learning for cross-project defect prediction," *Applied Soft Computing*, vol. 49, pp. 1062–1077, 2016.

[154] I. Arora and A. Saha, "Elm and kelm based software defect prediction using feature selection techniques," *Journal of Information and Optimization Sciences*, vol. 40, no. 5, pp. 1025–1045, 2019.

[155] W. Afzal and R. Torkar, "Towards benchmarking feature subset selection methods for software fault prediction," in *Computational intelligence and quantitative software engineering*. Springer, 2016, pp. 33–58.

[156] H. Alsghaier and M. Akour, "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier," *Software: Practice and Experience*, vol. 50, no. 4, pp. 407–427, 2020.

[157] S. Hosseini, B. Turhan, and M. Mäntylä, "Search based training data selection for cross project defect prediction," in *Proceedings of the the 12th international conference on predictive models and data analytics in software engineering*, 2016, pp. 1–10.

[158] T. Thaher and N. Arman, "Efficient multi-swarm binary harris hawks optimization as a feature selection approach for software fault prediction," in *2020 11th International conference on information and communication systems (ICICS)*. IEEE, 2020, pp. 249–254.

[159] S. Shukla, T. Radhakrishnan, K. Muthukumaran, and L. B. M. Neti, "Multi-objective cross-version defect prediction," *Soft Computing*, vol. 22, no. 6, pp. 1959–1980, 2018.

[160] W. Rhmann, B. Pandey, G. Ansari, and D. K. Pandey, "Software fault prediction based on change metrics using hybrid algorithms: An empirical study," *Journal of King Saud University-Computer and Information Sciences*, vol. 32, no. 4, pp. 419–424, 2020.

[161] N. Zhang, K. Zhu, S. Ying, and X. Wang, "Kaea: A novel three-stage ensemble model for software defect prediction." *Computers, Materials & Continua*, vol. 64, no. 1, 2020.

[162] F. S. Fazel, "A new method to predict the software fault using improved genetic algorithm," *Bulletin de la Société Royale des Sciences de Liège*, vol. 85, pp. 187–202, 2016.

[163] L. Kumar and S. K. Rath, "Application of genetic algorithm as feature selection technique in development of effective fault prediction model," in *2016 IEEE Uttar Pradesh Section International Conference on Electrical, Computer and Electronics Engineering (UPCON)*. IEEE, 2016, pp. 432–437.

[164] A. Oloduowo, M. Raheem, F. Ayinla, and B. Ayeyemi, "Software defect prediction using metaheuristic-based feature selection and classification algorithms," *Ilorin J Comput Sci Inf Technol*, vol. 3, pp. 23–39, 2020.

[165] C. Jin and E. Dong, "Software defect prediction using fuzzy integral and genetic algorithm," in *Software Engineering and Information Technology: Proceedings of the 2015 International Conference on Software Engineering and Information Technology (SEIT2015)*. World Scientific, 2016, pp. 334–340.

[166] R. B. Bahaweres, A. I. Suroso, A. W. Hutomo, I. P. Solihin, I. Hermadi, and Y. Arkeman, "Tackling feature selection problems with genetic algorithms in software defect prediction for optimization," in *2020 International Conference*

on Informatics, Multimedia, Cyber and Information System (ICIMCIS). IEEE, 2020, pp. 64–69.

[167] V. Jayaraj and N. S. Raman, "An hybrid multilayer perceptron using gso-ga for software defect prediction," in *Proc. of 2nd International Conf. on Recent Trends in Engineering Science and Management*, 2016, pp. 119–132.

[168] M. Banga, A. Bansal, and A. Singh, "Proposed approach to predict software faults detection using entropy," *International Journal of System Assurance Engineering and Management*, vol. 11, no. Suppl 2, pp. 301–312, 2020.

[169] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, "Software defect prediction using feature selection and random forest algorithm," in *2017 International conference on new trends in computing sciences (ICTCS)*. IEEE, 2017, pp. 252–257.

[170] R. Moussa and D. Azar, "A pso-ga approach targeting fault-prone software modules," *Journal of Systems and Software*, vol. 132, pp. 41–49, 2017.

[171] A. O. Balogun, S. Basri, S. A. Jadid, S. Mahamad, M. A. Al-momani, A. O. Bajeh, and A. K. Alazzawi, "Search-based wrapper feature selection methods in software defect prediction: an empirical analysis," in *Computer Science On-line Conference*. Springer, 2020, pp. 492–503.

[172] S. Hosseini, B. Turhan, and M. Mäntylä, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction," *Information and Software Technology*, vol. 95, pp. 296–312, 2018.

[173] D. Wu, J. Zhang, S. Geng, X. Cai, and G. Zhang, "A multi-objective bat algorithm for software defect prediction," in *International Conference on Bio-Inspired Computing: Theories and Applications*. Springer, 2019, pp. 269–283.

[174] A. Haveri and Y. Suresh, "Software fault prediction using artificial intelligence techniques," in *2017 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS)*. IEEE, 2017, pp. 1–5.

[175] M. Anbu and G. Anandha Mala, "Feature selection using firefly algorithm in software defect prediction," *Cluster Computing*, vol. 22, pp. 10 925–10 934, 2019.

[176] X. Rong and Z. Cui, "Hybrid algorithm for two-objective software defect prediction problem," *International Journal of Innovative Computing and Applications*, vol. 8, no. 4, pp. 207–212, 2017.

[177] R. Malhotra, N. Nishant, S. Gurha, and V. Rathi, "Application of particle swarm optimization for software defect prediction using object oriented metrics," in *2021 11th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, 2021, pp. 88–93.

[178] A. Pal, H. Jain, and M. Kumar, "Optimizing software error proneness prediction using bird mating algorithm," in *Software Project Management for Distributed Computing: Life-Cycle Methods for Developing Scalable and Reliable Tools*. Springer, 2017, pp. 257–287.

[179] M. Thangavel and R. Pugazendi, "Optimized support vector machine for software defect prediction," *Int J Eng Res Dev*, vol. 13, pp. 1–11, 2017.

[180] N. Zhang, S. Ying, W. Ding, K. Zhu, and D. Zhu, "Wgncs: A robust hybrid cross-version defect model via multi-objective optimization and deep enhanced feature representation," *Information Sciences*, vol. 570, pp. 545–576, 2021.

[181] J. Kang, S. Kwon, D. Ryu, and J. Baik, "Haspo: Harmony search-based parameter optimization for just-in-time software defect prediction in maritime software," *Applied sciences*, vol. 11, no. 5, p. 2002, 2021.

[182] Y. Cao, Z. Ding, F. Xue, and X. Rong, "An improved twin support vector machine based on multi-objective cuckoo search for software defect prediction," *International Journal of Bio-Inspired Computation*, vol. 11, no. 4, pp. 282–291, 2018.

[183] C. Jin, "Cross-project software defect prediction based on domain adaptation learning and optimization," *Expert Systems with Applications*, vol. 171, p. 114637, 2021.

[184] W. Rhmann, "Application of hybrid search based algorithms for software defect prediction," *International Journal of Modern Education and Computer Science*, vol. 12, no. 4, p. 51, 2018.

[185] Z. Li, T. Li, Y. Wu, L. Yang, H. Miao, and D. Wang, "Software defect prediction based on hybrid swarm intelligence and deep learning," *Computational Intelligence and Neuroscience*, vol. 2021, no. 1, p. 4997459, 2021.

[186] X. Yang, D. Lo, X. Xia, Y. Zhang, and J. Sun, "Deep learning for just-in-time defect prediction," in *2015 IEEE International conference on software quality, reliability and security*. IEEE, 2015, pp. 17–26.

[187] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *Proceedings of the 38th international conference on software engineering*, 2016, pp. 297–308.

[188] T. Ting, X.-S. Yang, S. Cheng, and K. Huang, "Hybrid metaheuristic algo-

rithms: past, present, and future," *Recent advances in swarm intelligence and evolutionary computation*, pp. 71–83, 2014.

[189] R. Hochman, T. M. Khoshgoftaar, E. B. Allen, and J. P. Hudepohl, "Using the genetic algorithm to build optimal neural networks for fault-prone module detection," in *Proceedings of ISSRE'96: 7th International Symposium on Software Reliability Engineering*. IEEE, 1996, pp. 152–162.

[190] M. Harman, "Why the virtual nature of software makes it ideal for search based optimization," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2010, pp. 1–12.

[191] S. Chulani, B. Boehm, and B. Steece, "Bayesian analysis of empirical software engineering cost models," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 573–583, 2002.

[192] B. Xue, M. Zhang, W. N. Browne, and X. Yao, "A survey on evolutionary computation approaches to feature selection," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 4, pp. 606–626, 2015.

[193] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, vol. 33, no. 1, pp. 2–13, 2006.

[194] M. D'Ambros, M. Lanza, and R. Robbes, "Evaluating defect prediction approaches: A benchmark and an extensive comparison," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 531–577, 2012.

[195] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: Recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT symposium*

and the 13th European conference on Foundations of software engineering. ACM, 2011, pp. 15–25.

[196] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415–1425, 1989.

[197] M. H. Halstead, *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., 1977.

[198] H. Turabieh, M. Mafarja, and X. Li, "Iterated feature selection algorithms with layered recurrent neural network for software fault prediction," *Expert Systems with Applications*, vol. 122, pp. 27–42, 2019.

[199] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[200] A. B. De Carvalho, A. Pozo, and S. R. Vergilio, "A symbolic fault-prediction model based on multiobjective particle swarm optimization," *Journal of Systems and Software*, vol. 83, no. 5, pp. 868–882, 2010.

[201] C. Catal, "Software fault prediction: A literature review and current trends," *Expert systems with applications*, vol. 38, no. 4, pp. 4626–4636, 2011.

[202] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[203] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[204] X. Chen, Y. Shen, Z. Cui, and X. Ju, "Applying feature selection to software defect prediction using multi-objective optimization," in *2017 IEEE 41st annual computer software and applications conference (COMPSAC)*, vol. 2. IEEE, 2017, pp. 54–59.

[205] A. B. de Carvalho, A. Pozo, S. Vergilio, and A. Lenz, "Predicting fault proneness of classes trough a multiobjective particle swarm optimization algorithm," in *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, vol. 2. IEEE, 2008, pp. 387–394.

[206] A. Sharma and P. K. Mishra, "Performance analysis of machine learning based optimized feature selection approaches for breast cancer diagnosis," *International Journal of Information Technology*, vol. 14, no. 4, pp. 1949–1960, 2022.

[207] R. Jayanthi and L. Florence, "Software defect prediction techniques using metrics based on neural network classifier," *Cluster Computing*, vol. 22, pp. 77–88, 2019.

[208] R. Malhotra, S. Chawla, and A. Sharma, "An artificial neural network model based on binary particle swarm optimization for enhancing the efficiency of software defect prediction," in *Proceedings of the 2023 6th International Conference on Software Engineering and Information Management*, 2023, pp. 92–100.

[209] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the royal statistical society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[210] G. Abaei and A. Selamat, "A survey on software fault detection based on

different prediction approaches," *Vietnam Journal of Computer Science*, vol. 1, no. 2, pp. 79–95, 2014.

[211] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[212] S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu, and D. Chen, "Fecar: A feature selection framework for software defect prediction," in *2014 IEEE 38th annual computer software and applications conference*. IEEE, 2014, pp. 426–435.

[213] H. S. Alamri, Y. A. Alsariera, and K. Z. Zamli, "Opposition-based whale optimization algorithm," *Advanced Science Letters*, vol. 24, no. 10, pp. 7461–7464, 2018.

[214] S. Kiranyaz, T. Ince, R. Hamila, and M. Gabbouj, "Convolutional neural networks for patient-specific ecg classification," in *2015 37th annual international conference of the IEEE engineering in medicine and biology society (EMBC)*. IEEE, 2015, pp. 2608–2611.

[215] R. Malhotra, S. Chawla, and A. Sharma, "Software defect prediction based on multi-filter wrapper feature selection and deep neural network with attention mechanism," *Neural Computing and Applications*, pp. 1–28, 2025.

[216] W. L. Wang, W. K. Li, Z. Wang, and L. Li, "Opposition-based multi-objective whale optimization algorithm with global grid ranking," *Neurocomputing*, vol. 341, pp. 41–59, 2019.

[217] B. Ghotra, S. McIntosh, and A. E. Hassan, "A large-scale study of the impact of feature selection techniques on defect classification models," in *2017*

IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, 2017, pp. 146–157.

[218] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical systems and signal processing*, vol. 151, p. 107398, 2021.

[219] O. Avci, O. Abdeljaber, S. Kiranyaz, B. Boashash, H. Sodano, and D. J. Inman, "Efficiency validation of one dimensional convolutional neural networks for structural damage detection using a shm benchmark data," in *Proc. 25th Int. Conf. Sound Vib.(ICSV)*, 2018, pp. 4600–4607.

[220] G. Sumbul, R. G. Cinbis, and S. Aksoy, "Multisource region attention network for fine-grained object recognition in remote sensing imagery," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 7, pp. 4929–4937, 2019.

[221] X. Qi, K. Li, P. Liu, X. Zhou, and M. Sun, "Deep attention and multi-scale networks for accurate remote sensing image segmentation," *IEEE Access*, vol. 8, pp. 146 627–146 639, 2020.

[222] W. Li, K. Liu, L. Zhang, and F. Cheng, "Object detection based on an adaptive attention mechanism," *Scientific Reports*, vol. 10, no. 1, p. 11307, 2020.

[223] Q. Hou, D. Zhou, and J. Feng, "Coordinate attention for efficient mobile network design," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 13 713–13 722.

[224] M. N. Uddin, B. Li, Z. Ali, P. Kefalas, I. Khan, and I. Zada, "Software defect prediction employing bilstm and bert-based semantic feature," *Soft Computing*, vol. 26, no. 16, pp. 7877–7891, 2022.

[225] G. Fan, X. Diao, H. Yu, K. Yang, and L. Chen, "Software defect prediction via attention-based recurrent neural network," *Scientific Programming*, vol. 2019, no. 1, p. 6230953, 2019.

[226] C. Pan, M. Lu, and B. Xu, "An empirical study on software defect prediction using codebert model," *Applied Sciences*, vol. 11, no. 11, p. 4793, 2021.

[227] R. Malhotra, S. Chawla, and A. Sharma, "Evaluating the effectiveness of meta-heuristic techniques for tuning svm parameters in software defect prediction," in *Intelligent Computing and Communication Techniques*. CRC Press, 2025, pp. 738–746.

[228] A. B. Farid, E. M. Fathy, A. S. Eldin, and L. A. Abd-Elmegid, "Software defect prediction using hybrid model (cbil) of convolutional neural network (cnn) and bidirectional long short-term memory (bi-lstm)," *PeerJ Computer Science*, vol. 7, p. e739, 2021.

[229] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *The journal of machine learning research*, vol. 13, no. 1, pp. 281–305, 2012.

[230] R. Malhotra, S. Chawla, and A. Sharma, "Enhancing software defect prediction through meta-heuristic hyperparameter optimization of ensemble learning models," *International Journal of System Assurance Engineering and Management*, 2025.

Supervisor's Biography

Prof. Ruchika Malhotra
Head of the Department & Professor
Department of Software Engineering
Delhi Technological University
Email: ruchikamalhotra@dtu.ac.in

Educational Qualifications:

Postdoc (Indiana University-Purdue University Indianapolis, USA), Ph.D (Computer Applications)

Ruchika Malhotra is the Head of the Department and a Professor in the Department of Software Engineering at Delhi Technological University (DTU), Delhi, India. She has previously served as the Associate Dean of Industrial Research and Development at DTU. She was awarded the prestigious Raman Fellowship for post-doctoral research at Indiana University-Purdue University Indianapolis, USA. She earned her master's and doctorate degrees in software engineering from the University School of Information Technology at Guru Gobind Singh Indraprastha University, Delhi, India. In 2013, she received the IBM Faculty Award. Her contributions to the field have earned her recognition as one of the world's top 2% scientists, according to a Stanford University report from 2020 to 2024, specifically for her work in "Artificial Intelligence & Image Processing" . She has also received the Commendable Research Award from Delhi Technological University for the years 2018-2024. Her h-index is 38 as reported by Google Scholar. She is the author of the book "Empirical Research in Software Engineering," published by CRC Press, and co-author of "Object Oriented Software Engineering," published by PHI Learning. She has published over 250 research papers in international journals and conferences. Her research interests include software testing, software quality improvement, statistical and adaptive prediction models, software metrics, and the definition and validation of software metrics.

Supervisor's Biography

Dr. Anjali Sharma
Head (IT) and Deputy Head
(Planning, Monitoring & Evaluation and ISTAG)
CSIR-NPL
Email: anjali1007@gmail.com

Educational Qualifications:
B.Tech(CSE), M.Tech(CSE) & Ph.D (Computer Science)

With over 20 years of experience across industry, academia, and national R&D, Dr. Anjali Sharma is a Senior Principal Scientist at CSIR–National Physical Laboratory (NPL), New Delhi, and Professor at AcSIR. She currently serves as Head (IT) and Deputy Head (Planning, Monitoring & Evaluation and ISTAG), overseeing international, externally funded, and internal research projects. She is actively involved in global digital metrology initiatives, contributing to CIPM Forum-MD task groups, PTB Germany's CABUREK program, and APMP's DXFG harmonization activities, and leads the Enterprise Architecture sub-group within the CIPM Metrology Semantics Task Group. She previously coordinated the national NPLONE program (2016–2024), enabling students across India to access advanced NPL research facilities. Her professional background spans program management, software development, ERP implementation, database administration, predictive modelling, and device commercialization, and she is co-inventor on patents granted in the US and Singapore.

Her research focuses on software quality, machine learning, predictive modelling, statistical data analysis, and digital metrology. She holds a PhD in Software Engineering from Delhi Technological University, an MTech in Computer Science from Banasthali Vidyapith, and a BTech in CSE. She is a member of BIS and APMP committees and has received the CSIR-NPL Outstanding Performance Award (2018) and the CSIR-ERP Project Champion Award (2013). She is also SCWCD5, SCJP, IBM DB2 Associate, and UGC-NET qualified.

Author's Biography

Sonali Chawla
Research Scholar
Department of Software Engineering
Delhi Technological University
Email: sonalichwl@gmail.com

Educational Qualifications:
B.Sc (H) Comp. Sc., MCA (SE)

Sonali Chawla received her B.Sc (H) degree in Computer Science from Keshav Mahavidyalaya, Delhi University, New Delhi, India, and MCA degree in Software Engineering from USICT, Guru Gobind Singh Indraprastha University Delhi, India. She has industrial experience of over 3.5 yrs and is also UGC-NET & JRF qualified. She is currently pursuing a Ph.D. in Software Engineering at Delhi Technological University. Her doctoral research focuses on the application of machine learning, deep learning and optimization techniques in various aspects of software quality. Her current research interests include Artificial Intelligence, software quality, predictive modelling, optimization techniques and data analytics.